

university of djilali bounaama



جامعة الجليلي بونامة

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
DJILALI BOUNAAMA UNIVERSITY - AIN DEFLA
FACULTY OF SCIENCES
COMPUTER SCIENCE DEPARTMENT

END OF STUDIES THESIS FOR OBTAINING THE MASTER'S
DEGREE IN COMPUTER SCIENCE

Option : " SOFTWARE ENGINEERING "

Theme :

Etude comparative des approches deep learning "
convolutional neural network " utilisées dans
l'optimisation des algorithmes FSM

AUTHORS:

- ✓ **ABADA YOUNES**
- ✓ **OURAG OKBA**

Under the supervision of the teacher :

➤ PROF O.HARBOUCHE

(superviseur)

october-2020

Abstract

Artificial intelligence still the topic of the present, that study a lot of life domains, till reach to the graphs ” FREQUENT SUB-GRAPH MINING “that use the approach of deep learning especially the graph convolutional network that deal with the graphs by a smart way, and apply some of its methods for reach to the more effecient result by a comparison.

الملخص

لا يزال الذكاء الاصطناعي هو موضوع المحاضر، الذي يدرس الكثير من مجالات الحياة، حتى الوصول إلى الرسوم البيانية» تعدين SUBGRAPH متكرر « التي تستخدم نهج التعلم العميق وخاصة شبكة الرسم البياني التلافيفية التي تتعامل مع الرسوم البيانية بطريقة ذكية، حيث تم تطبيق بعض أساليبها للوصول إلى النتيجة الأكثر فعالية من خلال المقارنة.

Résumé

L'intelligence artificielle est toujours le sujet du présent, qui étudie beaucoup de domaines de la vie, jusqu'à atteindre les graphiques " FREQUENT SUBGRAPH MINING " qui utilisent l'approche de l'apprentissage en profondeur en particulier le réseau convolutif de graphes qui traitent les graphiques de manière intelligente, et applique certaines de ses méthodes pour atteindre le résultat le plus efficace par une comparaison.

Acknowledgements

THERE ARE NO WORDS TO EXPRESS
OUR GRATITUDE TO PROF. O.HARBOUCHE,
OUR ADVISER WHO HELPED US TO IMPROVE NEARLY
EVERY ASPECT OF THIS CURRENT WORK, THE STUDY
PRESENTED IN THIS THESIS WOULD NOT HAVE
HAPPENED WITHOUT HIS SUPPORT, GUIDANCE, AND
ENCOURAGEMENT.

WE WOULD ALSO LIKE TO THANK OUR FAMILIES,
THEY WERE ALWAYS SUPPORTING AND
ENCOURAGING US WITH THEIR BEST WISHES.

Contents

Acknowledgements	I
List of Figures	VI
List of Tables	VIII
General Introduction	VIII
I Graph Theory and Frequent Subgraph Mining	1
I.1 data mining	2
I.2 Graph mining	3
I.3 What can we do with graph mining?	3
I.4 What is involved in graph mining?	4
I.5 Frequent subgraph mining (FSM)	4
I.5.1 Prerequisite	4
I.6 Types and properties of graphs	5
I.6.1 Finite Graphs	5
I.6.2 Infinite Graph	5
I.6.3 Trivial Graph	6
I.6.4 Simple Graph	6
I.6.5 Multi Graph	6
I.6.6 Null Graph	7
I.6.7 Complete Graph	7
I.6.8 Pseudo Graph	7
I.6.9 Regular Graph	8
I.6.10 Bipartite Graph	8
I.6.11 Labelled Graph	9
I.6.12 Digraph Graph	9
I.6.13 Subgraph	10
I.6.14 Types of Subgraph	10
I.6.15 Connected or Disconnected Graph	11
I.6.16 Cyclic Graph	11
I.6.17 Isomorphisme graph	11
I.6.18 Automorphism	12
I.6.19 Latice	12
I.6.20 Density	13
I.6.21 Trees	13
I.7 Tree Terminology	14
I.8 Overview of FSM	15
I.8.1 Graph isomorphism detection	16
I.8.2 Search strategy	17
I.8.3 FSM algorithmic approaches	17

I.8.3.1	Apriori Property	18
I.8.3.2	Apriori based approach	18
I.8.3.3	algorithme AprioriGraph	18
I.8.3.4	Pattern Growth Approach	19
I.8.3.5	algorithme PatternGrowthGraph	19
I.8.4	Comparison between	20
I.9	Other Applications	21
I.10	Conclusion	22
II	General Concept for Deep Learning	23
II.1	Introduction	24
II.2	Artificial Intelligence	24
II.2.1	What is Artificial Intelligence?	24
II.2.2	Philosophy of AI	24
II.2.3	Goals of AI	24
II.2.4	What Contributes to AI?	24
II.2.5	Programming Without and With AI	24
II.2.6	What is AI Technique?	24
II.2.7	Applications of AI	25
II.2.8	History of AI	27
II.2.9	What is Intelligence?	28
II.2.10	Types of Intelligence	28
II.2.11	What is Intelligence Composed of?	29
II.2.12	Difference between Human and Machine Intelligence	30
II.2.13	Real Life Applications of Research Areas	31
II.2.14	Task Classification of AI	32
II.3	Machine learning	34
II.3.1	Statistical Techniques	34
II.4	Machine Learning – Categories of Machine Learning	35
II.4.1	Supervised Learning	35
II.4.1.1	Regression	36
II.4.1.2	Classification	36
II.4.2	Unsupervised Learning	36
II.4.3	Reinforcement Learning	37
II.4.4	Deep Learning	38
II.4.5	Deep Reinforcement Learning	38
II.5	Machine Learning – Supervised Learning	38
II.5.1	Algorithms for Supervised Learning	38
II.5.2	k-Nearest Neighbours	38
II.5.3	Decision Trees	39
II.5.4	Naive Bayes	39
II.5.5	Logistic Regression	40
II.5.6	Machine Learning – Scikit-learn Algorithm	40
II.6	Machine Learning – Unsupervised Learning	41
II.6.1	Algorithms for Unsupervised Learning	41
II.6.2	k-means clustering	41
II.6.3	Cluster Identification	41
II.6.4	Machine Learning – Artificial Neural Networks	41
II.6.5	ANN Architectures	43
II.7	Machine Learning – Deep Learning	44
II.7.1	Applications	44
II.7.2	Untapped Opportunities of Deep Learning	44

II.7.3	What is Required for Achieving More Using Deep Learning?	44
II.7.4	Deep Learning -Disadvantages	44
II.7.5	Black Box approach	45
II.7.6	Duration of Development	45
II.7.7	Amount of Data	46
II.7.8	Computationally Expensive	46
II.7.9	Mathematical Notation	46
II.7.10	Probability Theory	46
II.7.11	Visualization	46
II.8	Machine Learning – Implementing Machine Learning	46
II.8.1	Language Choice	47
II.8.2	IDEs	47
II.8.3	Platforms	47
II.9	Conclusion	48
III	Deep Learning on Graphs	49
III.1	INTRODUCTION	50
III.2	NOTATIONS AND PRELIMINARIES	51
III.3	GRAPH RECURRENT NEURAL NETWORKS	51
III.3.1	Node-level RNNs	51
III.3.2	Graph-level RNNs	52
III.4	GRAPH CONVOLUTIONAL NETWORKS	53
III.4.1	Convolution Operations	53
III.4.1.1	Spectral Methods	53
III.4.1.2	The Efficiency Aspect	53
III.4.1.3	The Aspect of Multiple Graphs	53
III.4.1.4	Frameworks	54
III.4.2	Readout Operations	54
III.4.2.1	Statistics	55
III.4.2.2	Hierarchical Clustering	55
III.4.2.3	Imposing Orders and Others	55
III.4.2.4	Summary	55
III.4.3	Improvements and Discussions	55
III.4.3.1	Attention Mechanism	55
III.4.3.2	Residual and Jumping Connections	55
III.4.3.3	Edge Features	56
III.4.3.4	Sampling Methods	56
III.4.3.5	Inductive Setting	57
III.5	GRAPH AUTOENCODERS	58
III.5.1	Autoencoders	58
III.5.2	Variational Autoencoders	58
III.5.3	Improvements and Discussions	58
III.5.3.1	Adversarial Training	58
III.5.3.2	Inductive Learning	58
III.5.3.3	Similarity Measures	59
III.6	GRAPH REINFORCEMENT LEARNING	59
III.7	GRAPH ADVERSARIAL METHODS	59
III.7.1	Adversarial Training	60
III.7.2	Adversarial Attacks	60
III.8	DISCUSSIONS AND CONCLUSION	60
III.8.1	Applications	60
III.8.2	Implementations	60

III.8.3	Future Directions	61
III.9	Conclusion	61
IV	Implementation and Evaluation	62
IV.1	INTRODUCTION	63
IV.2	Implementation framework	63
IV.2.1	Python	63
IV.2.1.1	What is Python?	63
IV.2.1.2	It is used for	63
IV.2.1.3	What can Python do	63
IV.2.1.4	Why Python ?	63
IV.2.1.5	Good to know	64
IV.2.1.6	Python Syntax compared to other programming languages	64
IV.2.2	DATASETS	64
IV.2.2.1	CoRA Dataset	64
IV.2.2.2	citeseer dataset	64
IV.2.3	GOOGLE COLAB	64
IV.2.3.1	Colab definition	64
IV.2.3.2	How does colab work?	65
IV.2.3.3	What colab services are available?	65
IV.3	IMPLEMENTATION	65
IV.3.1	Semi-Supervised Classification with GCNs "Li et al" method	65
IV.3.1.1	dataset used	66
IV.3.1.2	Results Analysis	66
IV.3.1.3	Comparison with other methods	67
IV.3.2	Graph convolutional network: kiph and willing method	68
IV.3.2.1	SEMI-SUPERVISED NODE CLASSIFICATION	68
IV.3.2.2	GRAPH-BASED SEMI-SUPERVISED LEARNING	69
IV.3.2.3	EXPERIMENTS	69
IV.3.2.4	DATASETS	69
IV.3.2.5	RESULTS	69
IV.3.2.6	Results Analysis	69
IV.3.3	graph convolutional network : " LGCN method "	70
IV.3.3.1	introduction	70
IV.3.3.2	METHODS	70
IV.3.3.3	EXPERIMENTAL STUDIES	70
IV.3.3.4	Results Analysis	71
IV.3.3.5	abstract	72
IV.4	CONCLUSION	73
	General Conclusion	VII
	References	VIII
	List of Abbreviations	XI
	Appendices	XII

List of Figures

I.1	The steps of the KDD process	2
I.2	Graph mining Related Concept	3
I.3	Prerequisite	4
I.4	Finite Graphs	5
I.5	Finite Graphs	5
I.6	Finite Graphs	6
I.7	Simple Graphs	6
I.8	Multi Graph	6
I.9	Null Graph	7
I.10	Complete Graph	7
I.11	Pseudo Graph	7
I.12	Regular Graph	8
I.13	Bipartite Graph	8
I.14	Labelled Graph	9
I.15	Digraph Graph	9
I.16	Subgraph	10
I.17	Types of Subgraph	10
I.18	Connected or Disconnected Graph	11
I.19	Cyclic Graph	11
I.20	Isomorphisme Graph	12
I.21	Subgraph Isomorphism	12
I.22	Lattice(G)	13
I.23	TREE in general view	13
I.24	Single graph.	15
I.25	Frequent subgraphs	15
I.26	Subgraph Isomorphism.	16
I.27	Search strategy	17
I.28	Apriori based approach	18
II.1	Artificial intelligence is a science and technology	25
II.2	Intelligence Composed	29
II.3	Task Classificationof AI	32
II.4	Type of machine learning	35
II.5	Unsupervised Learning	37
II.6	Scikit-learn Algorithm cheat-sheet.	40
II.7	architecture neural network	42
II.8	A mostly complet chart of Neural Networks	43
II.9	neural network and it tells you that the image is of a dog	45
III.1	A categorization of deep learning methods on graphs	50

III.2 Different node sampling methods, in which the blue nodes indicate samples from one batch and the arrows indicate the sampling directions. The red nodes in (B) represent historical samples 56

.1 Take a copy code from github for kiph and welling method of GCNN. XIII

.2 Take a copy code from github for li et al method of GCNN. XIII

.3 Take a copy code from github for LGCN method of GCNN. XIV

.4 Import LGCN method code. XIV

.5 Import Kiph and Welling method code XIV

.6 Import Li et Al method code. XV

.7 LGCN run result. XV

.8 Kiph and welling run result XVI

.9 li et al run result. XVI

List of Tables

I.1	Categorization of exact matching (sub) graph isomorphism testing algorithms	16
I.2	The difference between FP growth and apriori algorithm	20
II.1	Programming Without and With AI	25
II.2	History of AI	27
II.3	Types of Intelligence	28
II.4	Reasoning types	29
II.5	Real Life Applications of Research Areas	31
II.6	Table of Task Classification of AI	33
II.7	Table of Task Classification of AI	33
III.1	Main Distinctions among Deep Learning Methods on Graphs	50
III.2	A table for Commonly Used Notations	51
III.3	The Main Characteristics of Graph Recurrent Neural Network (Graph RNNs)	52
III.4	A Comparison among Different Graph Convolutional Networks (GCNs). T.C. = Time Complexity, M.G. = Multiple Graphs	54
III.5	A Comparison among Different Graph Autoencoders (GAEs). T.C. = Time Complexity	58
III.6	The Main Characteristics of Graph Reinforcement Learning	59
III.7	The Main Characteristics of Graph Adversarial Methods	59
III.8	Libraries of Deep Learning on Graphs	60
IV.1	Dataset statistics	66
IV.2	Classification Accuracy On Cora	66
IV.3	Classification Accuracy on CiteSeer	67
IV.4	Accuracy under 20 Labels per Class	68
IV.5	Dataset statistics, as reported in Yang et al. (2016)	69
IV.6	Summary of results in terms of classification accuracy (in percent).	70
IV.7	Summary of datasets used in our experiments [30, 31]. The Cora, Citeseer, and Pubmed datasets are used for transductive learning experiments, while the PPI dataset is for inductive learning experiments. The degree attribute listed is the average node degree of each dataset, which helps the selection of the hyper-parameter k in LGCLs	71
IV.8	Results of transductive learning experiments for comparing the sub-graph training and whole-graph training strategies on the Cora, Citeseer, and Pubmed datasets. For comparison, we conduct experiments on LGCNs that employ the same whole-graph training strategy as GCNs, denoted as $LGCN_{whole}$	72
IV.9	Results of transductive learning experiments for comparing the LGCNsub and GCN layers on the Cora, Citeseer, and Pubmed datasets. Using the network architecture of LGCNsub, we replace LGCLs by GCN layers, resulting in the LGCNsub-GCN model	73
IV.10	Results of transductive learning experiments for comparing the $LGCN_{sub}$ and GCN layers on the Cora, Citeseer, and Pubmed datasets. Using the network architecture of $LGCN_{sub}$, we replace LGCLs by GCN layers, resulting in the $LGCN_{sub}$ -GCN model	73

General Introduction

We live in an era when almost everything around us produces some kind of data, a search is recorded on a search engine, a patient's heartbeat data is generated in the hospital, atoms are represented, social media is targeted, categorized and represented graphically, and therefore it must be stored somewhere. Organizations and institutions have been storing massive amounts of data of various types and shapes for several years now.

The data remains on backup tapes or drives, and that changes. Organizations now want to use this data to gain insight to help understand current problems, seize new opportunities and generate greater profits. The study and analysis of these volumes of data has a lot of advantage.

Being "bulky, high-dimensional, heterogeneous, complex, disorganized, incomplete, loud, and erratic" has spawned a term called Big Data.

Big Data is entering the field of artificial intelligence as a good resource to apply its methods such as the convolutional network in particular, as deep learning methods have become a source for solving most of the paradoxes encountered in various fields and it represents a smart way to deal with the graphs by which the huge data sets are represented in the parts of this graduation note.

Totally, the goal of our work is make an effective comparison study of deep learning approaches " graph convolutional network " used in the optimization of FSM algorithms, this studies can give a lot of solution in the big data problems and lets the deal with the big data so simple, easy and understood.

This work divided to:

first chapter: review each of the types and characteristics of graphs.

second chapter: the concept and elements of artificial intelligence, and from here we reach machine learning and deep learning.

third chapter: we talk in general about the various classifications of deep learning and about the principle of its work and its characteristics, and in particular we talk about the methods available in the categories.

forth chapter: choose the graph convolutional network category and take a three codes of three methods and compare their result.

Chapter I

Graph Theory and Frequent Subgraph Mining

Introduction

This Chapter defines in details the concept of graph mining and focus primarily on frequent subgraph mining (FSM), it is organized in two parts:

Part 01 mainly dedicated to present, in a simplified way, the basic notions related to graphs and graph theory.

Part 02, focus on presenting “the state of the art” of Frequent Subgraph mining (FSM) algorithms and techniques. A survey of current research in the field, and solutions to address the main research issues are also presented.

I.1 data mining

Data mining is a particular step in the process of Knowledge Discovery in databases (KDD), which has a to extract statistically significant and useful knowledge from a huge volume of raw data sets, extracting such important knowledge can be crucial, sometimes essential, for the next phase in the analysis: the modeling.[1]

The KDD process is outlined in figure I.1.[2]

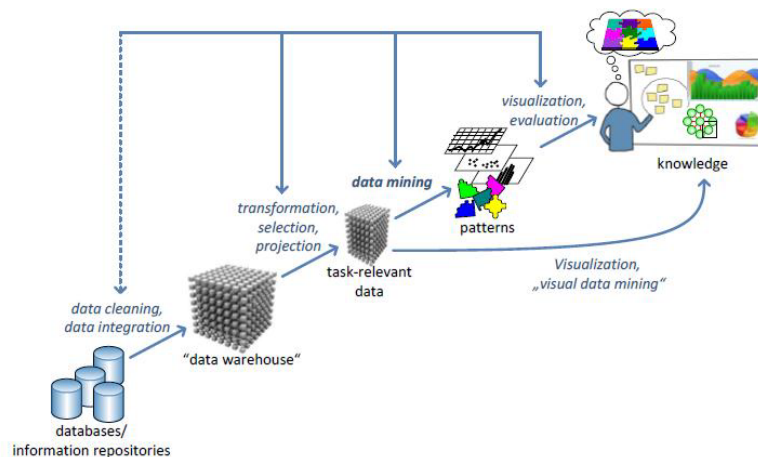


Figure I.1: The steps of the KDD process. [2]

The additional steps in the KDD process include the data selection and projection, and the visualization, and evaluation steps. During the past decade, the field of data mining has emerged as a novel field of research, investigating interesting research issues and developing challenging real-life applications. The objective data formats in the beginning of data mining, were limited to relational tables and transactions where each instance is represented by one row in a table or one transaction represented as a set. However, the studies within the last several years began to extend the classes of considered data to semi-structured data such as HTML and XML texts, symbolic sequences, and relations represented by advanced logics. Frequent pattern mining for instance has been a focused theme in data mining research for over a decade. Abundant literature has been dedicated to this research and tremendous progress has been made, ranging from efficient and scalable algorithms for frequent item set mining in transaction databases, mining association rules, to numerous research frontiers, such as sequential pattern mining. However, the arising and variety of the above-mentioned complex semi structured data and the need of discovering structural patterns in large datasets, which go beyond sets, sequences, and trees, toward complicated structure, makes the frequent item sets and frequent sequence mining approaches inefficient and unsuitable for such requirements, thus the emergence of graphs and frequent structural mining as a solution to these concerns. Certainly graphs as a data structure can meet the demands of modeling complicated substructure patterns and relations among data, and they are suitable representation for complex

objects so from this perspective, there has been much interest in the mining of graph data, (often referred to as graph based data mining or shortly graph mining).

I.2 Graph mining

Generally speaking, Graph mining is the process of discovering, retrieving and analyzing non trivial patterns in graph shaped data. Graph based data mining or graph mining has a strong relation with Multi-relational data mining. However, the main objective of graph mining is to provide new principles and efficient algorithms to mine topological substructures embedded in graph data, while the main objective of multi-relational data mining is to provide principles to mine and/or learn the relational patterns, represented by the expressive logical languages, the former is more geometry oriented and the latter more logic and relation oriented. I.2 [3]

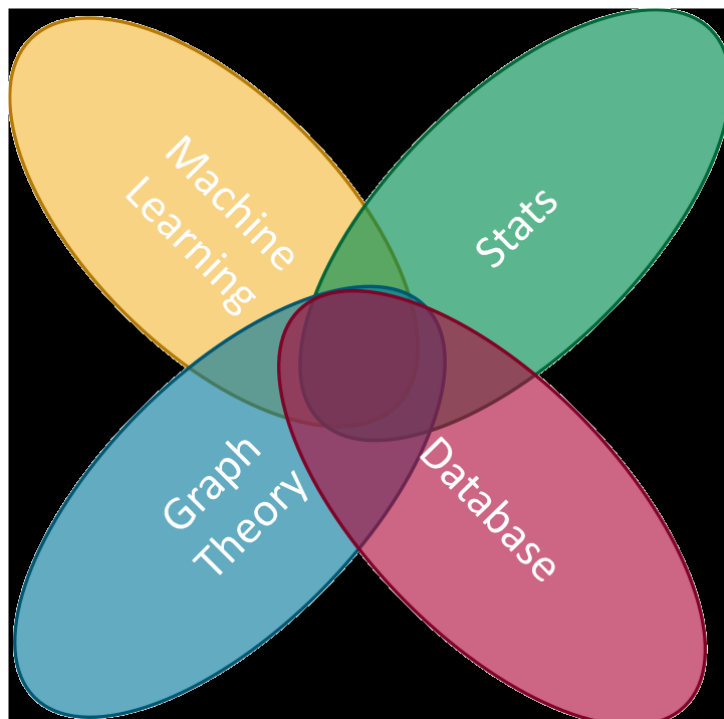


Figure I.2: Graph mining Related Concept. [3]

I.3 What can we do with graph mining?

- Compressing graphs without losing information
- Finding complex structures fast
- Recognizing communities and social patterns
- Study the propagation of viruses
- Predicting if two people will become friends
- Understanding what are the important nodes
- Showing how the network will evolve
- Helping the visualization of complex structures
- Finding roles, positive and negative influence prediction.[4]

I.4 What is involved in graph mining?

- Basic graph algorithms (shortest paths, BFS, isomorphisms, traversals, random walks ...)
- Storage and indexing
- Smart representations for compactness
- Modeling of problems as graphs
- Distance metrics and similarity measures
- Exact, Approximate, and heuristic algorithms
- Evolving structures
- Interactivity and online updates
- Complexity (most of the problems are not polynomially solvable). [4]

I.5 Frequent subgraph mining (FSM)

Among the various kinds of graph pattern, frequent subgraphs are very basic ones that can be discovered in a set of graphs (graph database) or a single large graph, they are useful at characterizing graphs sets, discriminating different groups of graphs, classifying and clustering graphs and building graphs indices, frequent subgraph mining encompass all the techniques and methodologies used to discover such patterns.

Before diving into the details of the variant frequent subgraph mining approaches and algorithms, let's review first some of graph theory basic concepts and terminologies.[4]

I.5.1 Prerequisite

A graph $G = (V, E)$ consists of a set of vertices $V = V_1, V_2, \dots$ and set of edges $E = E_1, E_2, \dots$. The set of unordered pairs of distinct vertices whose elements are called edges of graph G such that each edge is identified with an unordered pair (V_i, V_j) of vertices.

The vertices (V_i, V_j) are said to be adjacent if there is an edge E_k which is associated to V_i and V_j . In such a case V_i and V_j are called end points and the edge E_k is said to be connect/joint of V_i and V_j .I.3[5]

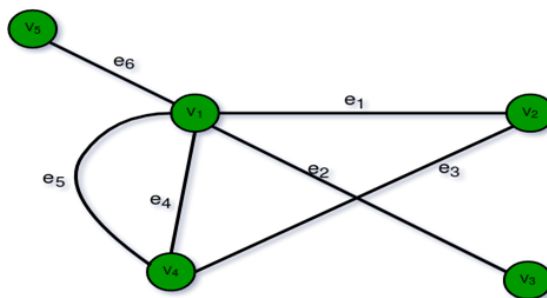


Figure I.3: Prerequisite. [5]

Preliminary definitions

In the following paragraphs a number of widely used definitions, used later in this chapter are introduced

I.6 Types and propertiefs of graphs

I.6.1 Finite Graphs

A graph is said to be finite if it has finite number of vertices and finite number of edges.(I.4) [5]

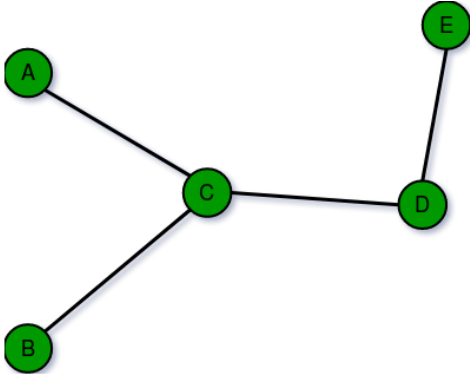


Figure I.4: Finite Graphs. [5]

I.6.2 Infinite Graph

A graph is said to be infinite if it has infinite number of vertices as well as infinite number of edges.(I.5) [5]

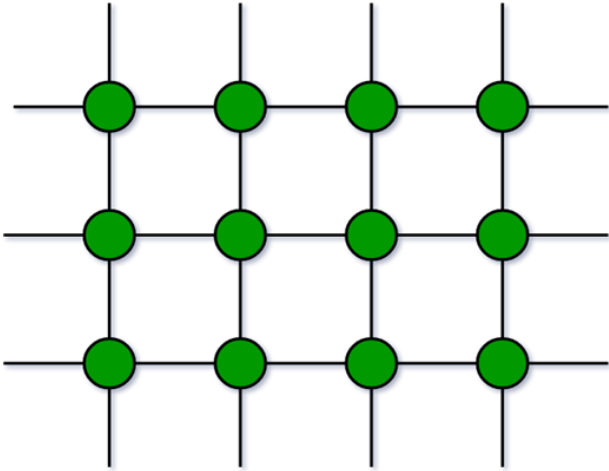


Figure I.5: Finite Graphs. [5]

I.6.3 Trivial Graph

A graph is said to be trivial if a finite graph contains only one vertex and no edge.(I.6) [5]

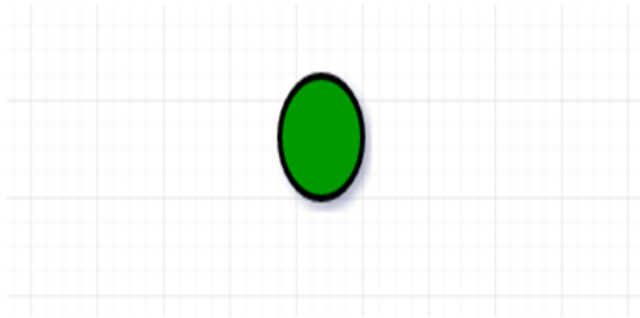


Figure I.6: Finite Graphs. [5]

I.6.4 Simple Graph

A simple graph is a graph which does not contains more than one edge between the pair of vertices. A simple railway tracks connecting different cities is an example of simple graph.(I.7) [5]



Figure I.7: Simple Graphs. [5]

I.6.5 Multi Graph

Any graph which contain some parallel edges but doesn't contain any self-loop is called multi graph. For example A Road Map.(I.8) [5]

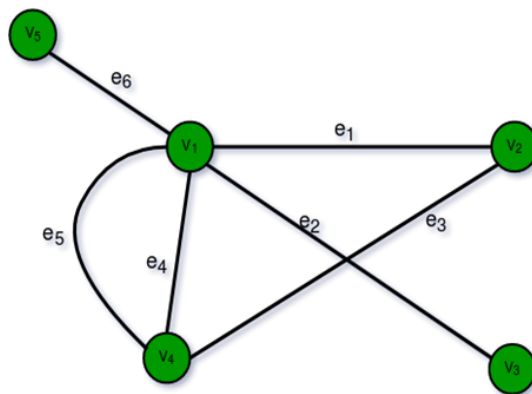


Figure I.8: Multi Graph. [5]

- **Parallel Edges:** If two vertices are connected with more than one edge than such edges are called parallel edges that is many roots but one destination.
- **Loop:** An edge of a graph which join a vertex to itself is called loop or a self-loop.

I.6.6 Null Graph

A graph of order n and size zero that is a graph which contains n number of vertices but do not contain any edge. (I.9) [5]

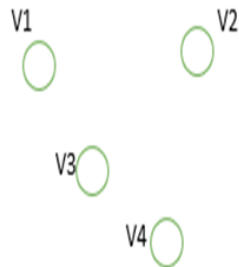


Figure I.9: Null Graph. [5]

I.6.7 Complete Graph

A simple graph with n vertices is called a complete graph if the degree of each vertex is $n-1$, that is, one vertex is attached with $n-1$ edges. A complete graph is also called Full Graph. (I.10) [5]



Figure I.10: Complete Graph. [5]

I.6.8 Pseudo Graph

A graph G with a self loop and some multiple edges is called pseudo graph. (I.11) [5]



Figure I.11: Pseudo Graph. [5]

I.6.9 Regular Graph

A simple graph is said to be regular if all vertices of a graph G are of equal degree. All complete graphs are regular but vice versa is not possible. (I.12) [5]

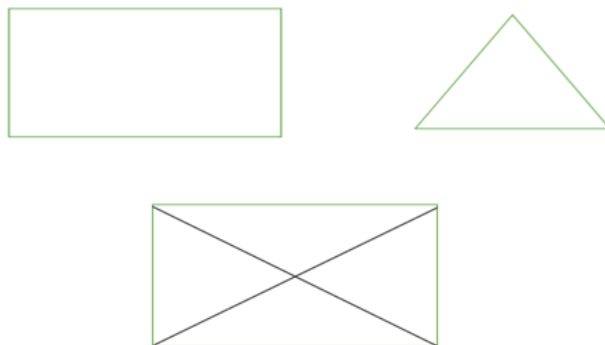


Figure I.12: Regular Graph. [5]

I.6.10 Bipartite Graph

A graph $G = (V, E)$ is said to be bipartite graph if its vertex set $V(G)$ can be partitioned into two non-empty disjoint subsets. $V_1(G)$ and $V_2(G)$ in such a way that each edge e of $E(G)$ has its one end in $V_1(G)$ and other end in $V_2(G)$.

The partition $V_1 \cup V_2 = V$ is called Bipartite of G . Here in the figure (I.13) :

$V_1(G) = V_5, V_4, V_3$

$V_2(G) = V_1, V_2$ [5]

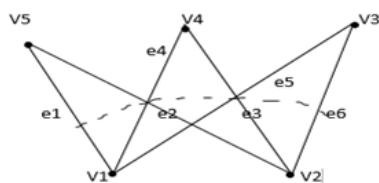


Figure I.13: Bipartite Graph. [5]

I.6.11 Labelled Graph

If the vertices and edges of a graph are labelled with name, data or weight then it is called labelled graph. It is also called *Weighted Graph*.(I.14) [5]

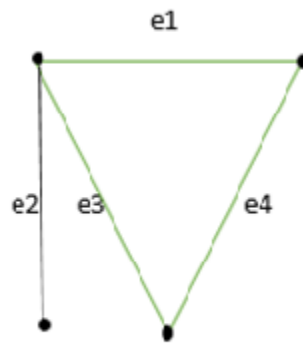


Figure I.14: Labelled Graph. [5]

I.6.12 Digraph Graph

A graph $G = (V, E)$ with a mapping f such that every edge maps onto some ordered pair of vertices (V_i, V_j) is called Digraph. It is also called Directed Graph. Ordered pair (V_i, V_j) means an edge between V_i and V_j with an arrow directed from V_i to V_j [5].

Here in the figure (I.15) : $e_1 = (V_1, V_2)$, $e_2 = (V_2, V_3)$, $e_4 = (V_2, V_4)$

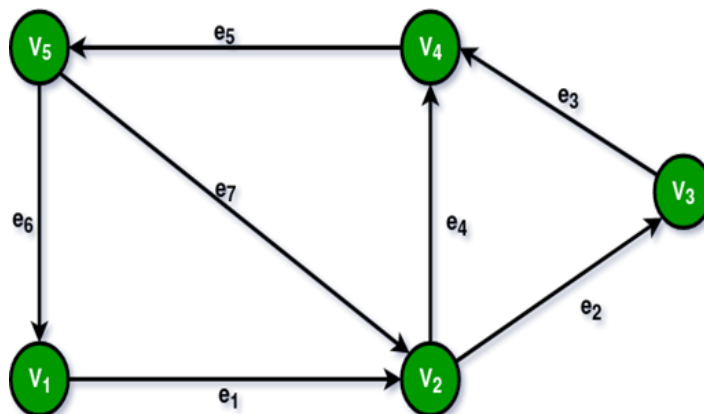


Figure I.15: Digraph Graph. [5]

I.6.13 Subgraph

A graph $G = (V_1, E_1)$ is called subgraph of a graph $G(V, E)$ if $V_1(G)$ is a subset of $V(G)$ and $E_1(G)$ is a subset of $E(G)$ such that each edge of G_1 has same end vertices as in G .(I.16)

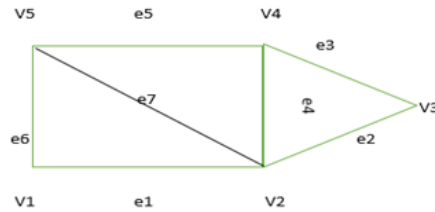


Figure I.16: Subgraph. [5]

I.6.14 Types of Subgraph

- **Vertex disjoint subgraph:** Any two graph $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are said to be vertex disjoint of a graph $G = (V, E)$ if $V_1(G_1) \cap V_2(G_2) = \text{null}$. In figure there is no common vertex between G_1 and G_2 .
 - **Edge disjoint subgraph:** A subgraph is said to be edge disjoint if $E_1(G_1) \cap E_2(G_2) = \text{null}$. In figure there is no common edge between G_1 and G_2 .
- Note:** Edge disjoint subgraph may have vertices in common but vertex disjoint graph cannot have common edge, so vertex disjoint subgraph will always be an edge disjoint subgraph[5].(I.17)

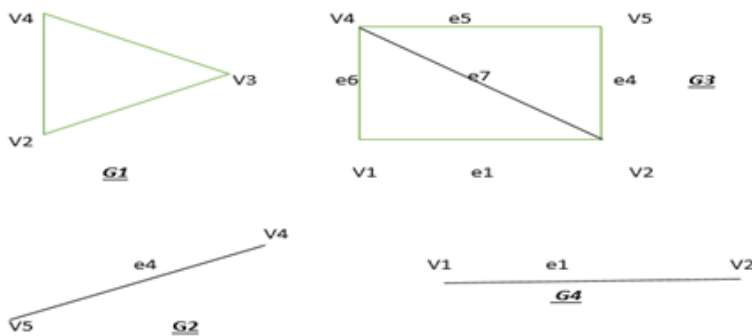


Figure I.17: Types of Subgraph. [5]

I.6.15 Connected or Disconnected Graph

A graph G is said to be connected if for any pair of vertices (V_i, V_j) of a graph G are reachable from one another. Or a graph is said to be connected if there exist atleast one path between each and every pair of vertices in graph G , otherwise it is disconnected. A null graph with n vertices is disconnected graph consisting of n components. Each component consist of one vertex and no edge.(I.18) [5]

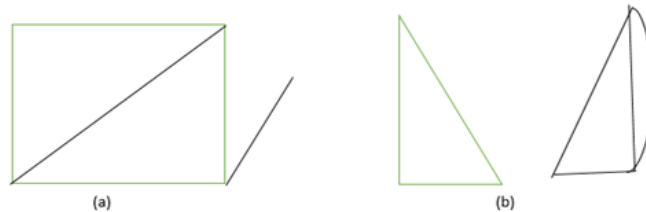


Figure I.18: Connected or Disconnected Graph. [5]

I.6.16 Cyclic Graph

A graph G consisting of n vertices and $n \geq 3$ that is $V_1, V_2, V_3, \dots, V_n$ and edges $(V_1, V_2), (V_2, V_3), (V_3, V_4), \dots, (V_n, V_1)$ are called cyclic graph.(I.19) [5]

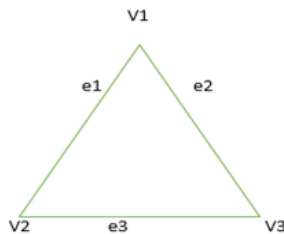


Figure I.19: Cyclic Graph. [5]

I.6.17 Isomorphisme graph

Isomorphism is a very general concept that appears in several areas of mathematics. The word derives from the Greek iso, meaning yequal," and morphosis, meaning "to form" or "to shape."

Two graphs are isomorphic, if they are structurally identical, Which means that they correspond in all structural details. Formal vertex-to-vertex and edge -to-edge correspondence is called isomorphism.

Two graph are said to be isomorphic if They [6]

- have the same no of vertices.
- They have the same number of edges.
- They have an equal number of vertices with a given degree.

Two Figure present respectively graph and subgraph isomorphism: (I.20)



Figure I.20: Isomorphisme Graph. [6]

The two graphs shown above are isomorphic, despite their different looking drawings.(I.21)

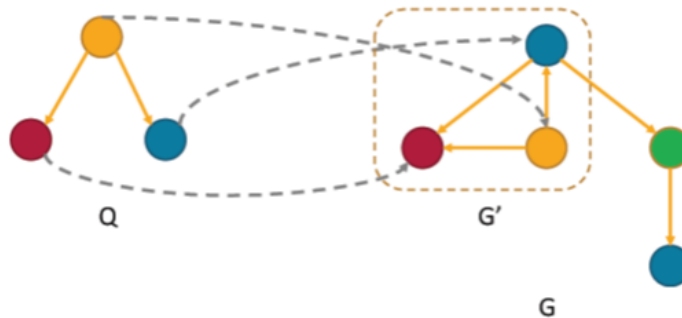


Figure I.21: Subgraph Isomorphism. [6]

I.6.18 Automorphism

An isomorphic mapping of the vertices of a graph G onto themselves (which also preserves the adjacency relationship) is called an automorphism of a graph G .

Evidently, each graph possesses a trivial automorphism which is called the identity automorphism. For some graphs, it is the only automorphism; these are called identity graphs. The set of all automorphisms of a graph G forms a group which is called the automorphism group of G . [7]

I.6.19 Lattice

Given a database G , a lattice is a structural form used to model the search space for finding frequent subgraphs, where each vertex represents a connected subgraph of the graph in G . The lowest vertex depicts the empty subgraph and the vertexes at the highest level depict the graphs in G . A vertex p is a parent of the vertex q in the lattice, if q is a subgraph of p , and q is different from p by exactly one edge. The vertex q is a child of p . All the subgraphs of each graph $G_i \in G$ which occur in the database are present in the lattice and every subgraph occurs only once in it. (I.22) [6]

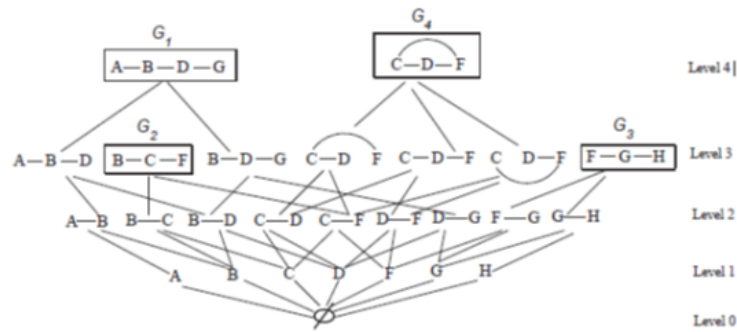


Figure I.22: Lattice(G). [5]

Example:

given a graph data set $G = G_1, G_2, G_3, G_4$, the corresponding Lattice(G), is given in Figure II-16. In the figure, the lowest vertex \emptyset represents the empty subgraph, and the vertexes at the highest level correspond to G_1, G_2, G_3 , and G_4 . The parents of the subgraph $B-D$ are subgraphs $A-B-D$ (joining the edge $A-B$) and $B-D-G$ (joining the edge $D-G$). Similarly, subgraphs $B-C$ and $C-F$ are the children of the subgraph $B-C-F$. (I.22)

I.6.20 Density

The density of a graph $G = (V; E)$ is calculated by: $density(G) = \frac{2 \cdot |E|}{|V| \cdot (|V| - 1)}$. The graph density measures the ratio of the number of edges compared to the maximal number of edges in a complete graph. A graph is said to be dense if the ratio is close to 1, and is considered as sparse if the ratio is close to 0. [8]

I.6.21 Trees

Trees are non-linear or hierarchal data structure unlike linked lists and arrays. Those are linear data structures. You can see an example of this in the picture of a tree structure above. Trees are in-fact a special type of graph with only one way from point A to point B. They are a connect of nodes which are connected through edges. Each node contains a value in addition they may or may not have a child node. (I.23)

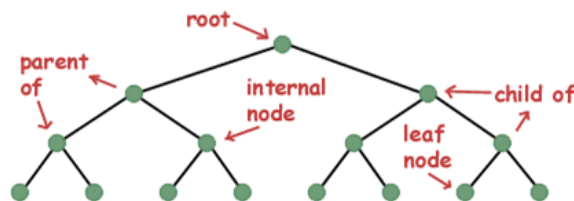


Figure I.23: TREE in general view .

I.7 Tree Terminology

In a tree data structure, we use the following terminology... [9]

Root In a tree data structure, the first node is called as Root Node. Every tree must have a root node. We can say that the root node is the origin of the tree data structure.

Edge In a tree data structure, the connecting link between any two nodes is called as EDGE.

Child In a tree data structure, the node which is descendant of any node is called as CHILD Node. In simple words, the node which has a link from its parent node is called as child node.

Parent In a tree data structure, the node which is a predecessor of any node is called as PARENT NODE.

Leaf In a tree data structure, the node which does not have a child is called as LEAF Node. In simple words, a leaf is a node with no child.

Height In a tree data structure, the total number of edges from leaf node to a particular node in the longest path is called as HEIGHT of that Node.

Depth In a tree data structure, the total number of edges from root node to a particular node is called as DEPTH of that Node.

Level In a tree data structure, the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on.

Sub Tree In a tree data structure, each child from a node forms a subtree recursively. Every child node will form a subtree on its parent node.

Path In a tree data structure, the sequence of Nodes and Edges from one node to another node is called as PATH between that two Nodes.

Degree In a tree data structure, the total number of children of a node is called as DEGREE of that Node.

Internal Nodes In a tree data structure, the node which has atleast one child is called as INTERNAL Node. In simple words, an internal node is a node with atleast one child.

Siblings In a tree data structure, nodes which belong to same Parent are called as SIBLINGS. In simple words, the nodes with the same parent are called Sibling nodes.

I.8 Overview of FSM

Besides discovering graphs common to several graphs, there is also a variation of the problem of frequent subgraph mining that consists of **finding all frequent subgraphs in a single graph** rather than in a graph database. The idea is almost the same. The goal is also to discover subgraphs that appear frequently or that are interesting. The only difference is how the support (frequency) is calculated. For this variation, the support of a subgraph is the number of times that it appears in the single input graph. For example, consider the following input graph(I.24).

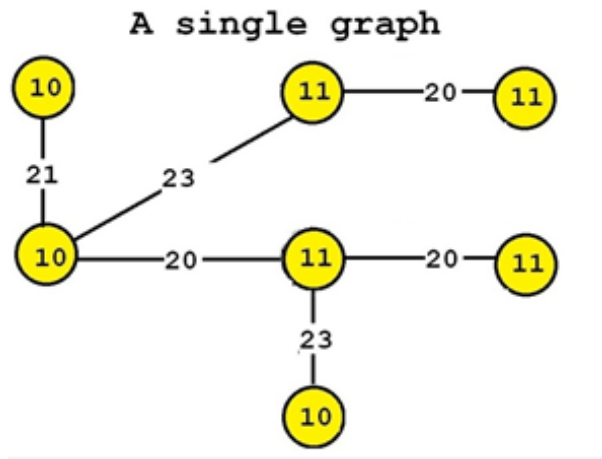


Figure I.24: Single graph.

This graph contains seven vertices and six edges. If we perform frequent subgraph mining on this single graph by setting the minsup parameter to 2, we can discover the five following frequent subgraphs:(I.25)

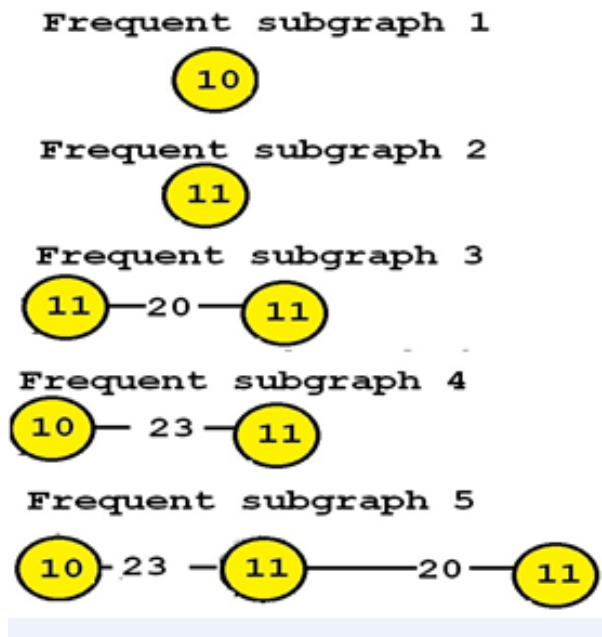


Figure I.25: Frequent subgraphs .

These subgraphs are said to be frequent because they appear at least twice in the input graph. For example, consider “Frequent subgraph 5”. This subgraph has a support of 2 because it has two occurrences in the input graph. Those two occurrences are highlighted below in red and blue, respectively.(I.26)

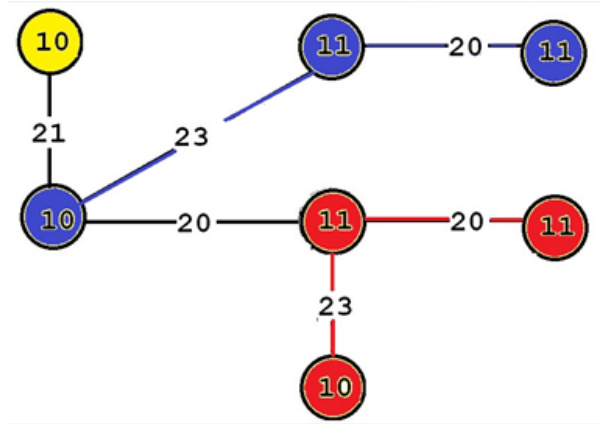


Figure I.26: Subgraph Isomorphism.

I.8.1 Graph isomorphism detection

The kernel of FSM is (sub)graph isomorphism detection. Graph isomorphism is neither known to be solvable in polynomial time nor NP-complete, while subgraph isomorphism, where we wish to establish whether a subgraph is wholly contained within a super graph, is known to be NP-complete . When restricting the graphs to trees, (sub)graph isomorphism detection becomes (sub)tree isomorphism detection. Tree isomorphism detection can be solved in a linear time. Subgraph isomorphism detection is fundamental to FSM. A significant number of “efficient” techniques have been proposed, all directed at reducing, as far as possible, the computational overhead associated with it. Subgraph isomorphism detection techniques can be roughly categorized as being either: exact matching or error tolerant matching . Most FSM algorithms adopt exact matching. A categorization of the main exact matching subgraph isomorphism detection algorithms is presented in Table (I.1) .

Algorithhme	Main Techniques	Mathing Types
Ullman	Backtracking + look ahead	Graph and subgraph isomorphism
SD	Distance matrix + backtracking	Graph isomorphism
Nauty	Group theory + canonical labeling	Graph isomorphism

Table I.1: Categorization of exact matching (sub) graph isomorphism testing algorithms. [10]

With reference to Table (I.1), Ullmann’s algorithm employs a backtracking procedure with a look-ahead function to reduce the size of the search space [10] . The SD algorithm, in turn, utilizes a distance matrix representation of a graph with a backtracking procedure to reduce the search [11] .

The Nauty algorithm [10] uses group theory to transform graphs to be matched into a canonical form so as to provide for more efficient and effective graph isomorphism checking. However, it has been noted [11] that the construction of the canonical forms can lead to exponential complexity in the worst case. Although Nauty was regarded as the fastest graph isomorphism algorithm by Conte [11] , Miyazaki in [12] demonstrated that there exist some categories of graphs which required exponential time to generate the canonical labelling. The **VF** [13] and

VF2 [12] use a Depth First Search (DFS) strategy, assisted by a set of feasibility rules to prune the search tree. VF2 is an improved version of VF, that explores the search space more effectively so that the matching time and the memory consumption are significantly reduced.

I.8.2 Search strategy

there is two kind of search basic strategies used for mining frequent subgraph ,the DFS and the BFS [17]

The breadth-first search (BFS) strategy The BFS strategy checks the support of all candidates of a certain size, before moving to the next level; i.e. first all possible candidate subgraphs of size k will be generated and checked for support, subsequently the frequent subgraphs will be retained and used to generate the candidate subgraphs of size $(k+1)$. A BFS is necessary if the subgraph candidates are generated by the join-based generation method. For example, to generate the $(k+1)$ -size subgraph candidates two k -size frequent graphs are needed, which means that all frequent subgraphs of size k need to be determined first. This approach accounts for effective candidate pruning, but at the cost of a high memory usage.I.27

The depth-first search (DFS) strategy The DFS strategy first checks the support of a candidate subgraph of size k ; if this subgraph is frequent it will be extended to size $(k+1)$ and checked for support again (I.27). The subgraph will continue to be extended until it is no longer frequent. Compared to the BFS this approach requires less memory but at the cost of less effective pruning.

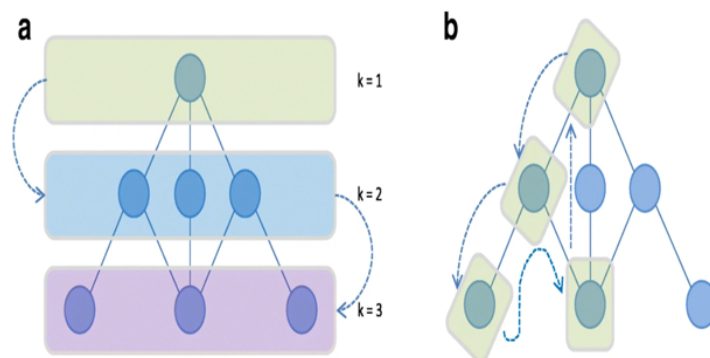


Figure I.27: Search strategy. [17]

Search strategy. a Breadth-first search (BFS), which will exhaust all relevant candidate subgraphs of a given size before proceeding to the next size. Candidate subgraphs of a larger size are then generated based on joining two subgraphs of a smaller size. b Depth-first search (DFS), which will explore an entire branch of the subgraph lattice before restarting at the top with a different branch. New candidate subgraphs are extended following a fixed set of rules until it drops below the frequency threshold or has reached a predefined maximum size.

I.8.3 FSM algorithmic approaches

There are two approaches by which frequent sub graphs can be found in a given graph / database.

- * Apriori – based approach
- * Pattern-growth approach

These two categories are similar in spirit to counterparts found in Association Rule Mining (ARM), namely the Apriori algorithm and FP-growth algorithm respectively. Before defining each of them let's review first the apriority property.

I.8.3.1 Apriori Property

The Apriori property also known as the downward closure property (DCP), expresses a monotonic decrease of an evaluation criterion accompanying with the progress of a sequential pattern mining. It is activated in order to efficiently discover all frequent sequential patterns, in simple terms this property impose that if a graph is frequent, then all of its subgraphs will also be frequent, thus the frequency or the support of a sequential graph is always decreasing or remaining constant, and never increases to overpass the support of its parent subgraphs.

This property must be hold for both the Apriori and the Pattern growth based algorithms to safely prune the candidates that are not frequent.[33]

I.8.3.2 Apriori based approach

- The Apriori property (anti-monotonicity): A size-k subgraph is frequent if and only if all of its subgraphs are frequent [1]
- A candidate size-(k+1) edge/vertex subgraph is generated if its corresponding two k-edge/vertex subgraph are frequent
- Iterative mining process:
 - Candidate-generation → candidate pruning → support counting → candidate elimination. [18]

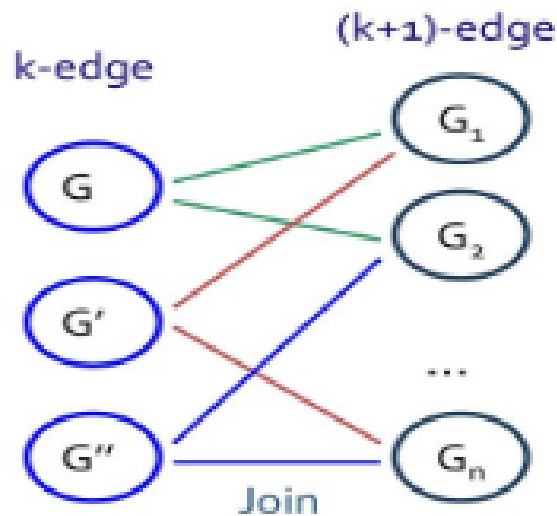


Figure I.28: Apriori based approach. [18]

I.8.3.3 algorithm AprioriGraph

AprioriGraph

- Level wise mining method
- Size of new substructures is increased by 1
- Generated by joining two similar but slightly different frequent sub-graphs
- Frequent is then checked.[18]

Algorithm: AprioriGraph. Apriori-based frequent substructure mining.

Input:

- D , a graph data set;
- min_sup , the minimum support threshold.

Output:

- S_k , the frequent substructure set.

Method:

$S_1 \leftarrow$ frequent single-elements in the data set;
Call AprioriGraph(D, min_sup, S_1);

procedure AprioriGraph(D, min_sup, S_k)

- (1) $S_{k+1} \leftarrow \emptyset$;
- (2) for each frequent $g_i \in S_k$ do
- (3) for each frequent $g_j \in S_k$ do
- (4) for each size $(k + 1)$ graph g formed by the merge of g_i and g_j do
- (5) if g is frequent in D and $g \notin S_{k+1}$ then
- (6) insert g into S_{k+1} ;
- (7) if $S_{k+1} \neq \emptyset$ then
- (8) AprioriGraph(D, min_sup, S_{k+1});
- (9) return;

I.8.3.4 Pattern Growth Approach

1. Initially, start with the frequent vertices as frequent graphs.
2. Extend these graphs by adding a new edge such that newly formed graphs are frequent graphs.
3. for each discovered graph g, k it performs extensions recursively until all the frequent graphs with g embedded are discovered.
4. The recursion stops once no frequent graph can be generated.[19]

I.8.3.5 algorithm PatternGrowthGraph

Simplistic Pattern Growth-based Frequent Substructure Mining [19]

mining. **Input:**

- g , a frequent graph;
- D , a graph data set;
- min_sup , minimum support threshold.

Output:

- The frequent graph set S .

Method:

- (1) if $g \in S$ then return;
- (2) else insert g into S ;
- (3) scan D once, find all the edges e such that g can be extended to $g \diamond_x e$;
- (4) for each frequent $g \diamond_x e$ do
 - (5) $PatternGrowthGraph(g \diamond_x e, D, min_sup, S)$;
- (6) return;

I.8.4 Comparison between

Comparison between FP growth algorithm and Apriori algorithm. [20]

FP growth algorithm	Apriori algorithm
FP growth algorithm is faster than Apriori algorithm.	It is slower than FP growth algorithm.
FP growth algorithm is an array based algorithm.	Apriori algorithm is a tree-based algorithm.
FP growth algorithm required only two database scan.	It requires multiple database scan to generate a candidant set.
It uses depth-first search.	It uses breadth-furst serch.

Table I.2: The difference between FP growth and apriori algorithm. [20]

I.9 Other Applications

Apart from Graph classification, Clustering and indexing FSM has many applications in interdisciplinary research such as chemical informatics , bioinformatics .[21] For example, Maximal subgraph mining is used in discovering structure motifs in a graph of homology protein where they encode the maximal structure commonalities within the group. It also has a cardinal influence in social networking and ego networks as well. Below tabular column shows a list of FSM algorithms in a 2D matrix form in which ith row and jth column has a value x.

(ith row) i – denotes the name of the Algorithm

(jth row) j – candidate generation method

x – candidate graph representation.

<i>Approach</i>	<i>Algorithm</i>	<i>Candidate Generation</i>			
		Level-Wise join	Right-most path Extn	Extension and join	Paths, trees and Graphs- Enumeratin
A Priori	FSG	Adjacency List	-	-	-
	AGM	CAM	-	-	-
	HSIGRAM	CAM	-	-	-
	FFSM	-	-	CAM	-
Pattern-growth	gSpan	-	Adjacency List	-	-
	Gaston	-	-	-	Hash Table
	MoFa	-	-	Embedding List	-
	SUBDUE	CAM	-	-	-
	CloseGraph	-	Adjacency List	-	-

I.10 Conclusion

On the work above we introduced the chapter by defining graph mining, then we gave some preliminary notions of graphs and graph theory. After that we focused in this chapter on Frequent Subgraph Mining (FSM), and we tried to give the major operations that an FSM algorithm has to perform in order to find subgraphs, for instance candidate generation, isomorphism checking and support counting and presented the techniques associated to perform each of them, finally we presented the FSM algorithmic approaches, Apriori and Pattern Growth.

Chapter II

General Concept for Deep Learning

II.1 Introduction

What used to be just a pipe dream in the realms of science fiction, artificial intelligence (AI) is now mainstream technology in our everyday lives with applications in image and voice recognition, language translations, chatbots, and predictive data analysis. In this part, we'll introduce AI along with its related terms machine learning and deep learning. By the end of the part you should understand these terms, how things generally work.[34]

II.2 Artificial Intelligence

II.2.1 What is Artificial Intelligence?

According to the father of Artificial Intelligence John McCarthy, it is “The science and engineering of making intelligent machines, especially intelligent computer programs”. Artificial Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner the intelligent humans think. AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems [22].

II.2.2 Philosophy of AI

While exploiting the power of the computer systems, the curiosity of human, lead him to wonder, “Can a machine think and behave like humans do?” Thus, the development of AI started with the intention of creating similar intelligence in machines that we find and regard high in humans [22].

II.2.3 Goals of AI

To Create Expert Systems: The systems which exhibit intelligent behavior, learn,[22]

- Demonstrate, explain, and advice its users. To Implement Human Intelligence in Machines: Creating systems that.
- Understand, think, learn, and behave like humans.

II.2.4 What Contributes to AI?

Artificial intelligence is a science and technology based on disciplines such as Computer Science, Biology, Psychology, Linguistics, Mathematics, and Engineering. A major thrust of AI is in the development of computer functions associated with human intelligence, such as reasoning, learning, and problem solving. Out of the following areas, one or multiple areas can contribute to build an intelligent system [22].(II.1)

II.2.5 Programming Without and With AI

The programming without and with AI is different in following ways:(II.1) [3]

II.2.6 What is AI Technique?

In the real world, the knowledge has some unwelcomed properties:

- * Asterisk Its volume is huge, next to unimaginable.

<i>Programming Without AI</i>	<i>Programming With AI</i>
A computer program without AI can answer the specific questions it is meant to solve.	A computer program with AI can answer the generic questions it is meant to solve.
Modification in the program leads to change in its structure.	AI programs can absorb new modifications by putting highly independent pieces of information together. Hence you can modify even a minute piece of information of program without affecting its structure.
Modification is not quick and easy. It may lead to affecting the program adversely.	Quick and Easy program modification.

Table II.1: Programming Without and With AI. [3]

- * Asterisk It is not well-organized or well-formatted.
- * Asterisk It keeps changing constantly.

AI Technique is a manner to organize and use the knowledge efficiently in such a way that:

- * It should be perceivable by the people who provide it.
- * It should be easily modifiable to correct errors.
- * It should be useful in many situations though it is incomplete or inaccurate.

AI techniques elevate the speed of execution of the complex program it is equipped with.[22]

II.2.7 Applications of AI

AI has been dominant in various fields such as [22]:

- **Gaming:** AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge. Natural Language .

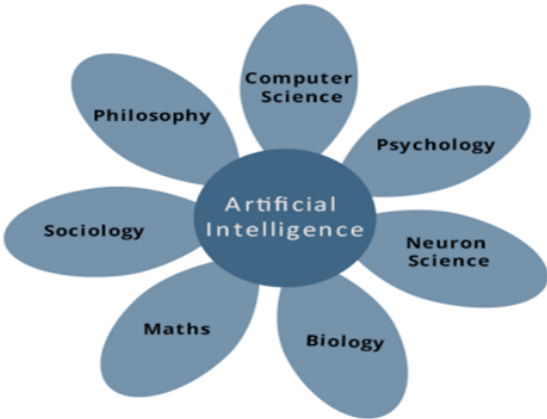


Figure II.1: Artificial intelligence is a science and technology. [22]

- **Processing:**It is possible to interact with the computer that understands natural language spoken by humans.
- **Expert Systems:**There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users .
- **Vision Systems:**These systems understand, interpret, and comprehend visual input on the computer. For example,
 - A spying aeroplane takes photographs which are used to figure out spatial information or map of the areas.
 - Doctors use clinical expert system to diagnose the patient.
 - Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.
- **Speech Recognition:**Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.
- **Handwriting Recognition :** The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text .
- **Intelligent Robots:** Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment .

II.2.8 History of AI

Here is the history of AI during 20th century:[22]

<i>Year</i>	<i>Milestone / Innovation</i>
1923	Karel Čapek play named “Rossum’s Universal Robots” (RUR) opens in London, first use of the word "robot" in English.
1943	Foundations for neural networks laid.
1945	Isaac Asimov, a Columbia University alumni, coined the term Robotics.
1950	Alan Turing introduced Turing Test for evaluation of intelligence and published Computing Machinery and Intelligence. Claude Shannon published Detailed Analysis of Chess Playing as a search.
1956	John McCarthy coined the term Artificial Intelligence. Demonstration of the first running AI program at Carnegie Mellon University.
1958	John McCarthy invents LISP programming language for AI.
1964	Danny Bobrow’s dissertation at MIT showed that computers can understand natural language well enough to solve algebra word problems correctly.
1965	Joseph Weizenbaum at MIT built ELIZA, an interactive problem that carries on a dialogue in English.
1969	Scientists at Stanford Research Institute Developed Shakey, a robot, equipped with locomotion, perception, and problem solving.
1973	The Assembly Robotics group at Edinburgh University built Freddy, the Famous Scottish Robot, capable of using vision to locate and assemble models.
1979	The first computer-controlled autonomous vehicle, Stanford Cart, was built.
1985	Harold Cohen created and demonstrated the drawing program, Aaron.
1990	Major advances in all areas of AI – <ul style="list-style-type: none"> • Significant demonstrations in machine learning • Case-based reasoning • Multi-agent planning • Scheduling • Data mining, Web Crawler • natural language understanding and translation • Vision, Virtual Reality • Games
1997	The Deep Blue Chess Program beats the then world chess champion, Garry Kasparov.
2000	Interactive robot pets become commercially available. MIT displays Kismet, a robot with a face that expresses emotions. The robot Nomad explores remote regions of Antarctica and locates meteorites.

Table II.2: History of AI. [22]

While studying artificially intelligence, you need to know what intelligence is. This chapter covers Idea of intelligence, types, and components of intelligence.

II.2.9 What is Intelligence?

The ability of a system to calculate, reason, perceive relationships and analogies, learn from experience, store and retrieve information from memory, solve problems, comprehend complex ideas, use natural language fluently, classify, generalize, and adapt new situations.[22]

II.2.10 Types of Intelligence

As described by Howard Gardner, an American developmental psychologist, the Intelligence comes in multifold[22]:

<i>Intelligence</i>	<i>Description</i>	<i>Example</i>
Linguistic intelligence	The ability to speak, recognize, and use mechanisms of phonology (speech sounds), syntax (grammar), and semantics (meaning).	Narrators, Orators
Musical intelligence	The ability to create, communicate with, and understand meanings made of sound, understanding of pitch, rhythm.	Musicians, Singers, Composers
Logical-mathematical intelligence	The ability of use and understand relationships in the absence of action or objects. Understanding complex and abstract ideas.	Mathematicians, Scientists
Spatial intelligence	The ability to perceive visual or spatial information, change it, and re-create visual images without reference to the objects, construct 3D images, and to move and rotate them.	Map readers, Astronauts, Physicists
Bodily-Kinesthetic intelligence	The ability to use complete or part of the body to solve problems or fashion products, control over fine and coarse motor skills, and manipulate the objects.	Players, Dancers
Intra-personal intelligence	The ability to distinguish among one's own feelings, intentions, and motivations.	Gautam Buddha
Interpersonal intelligence	The ability to recognize and make distinctions among other people's feelings, beliefs, and intentions.	Mass Communicators, Interviewers

Table II.3: Types of Intelligence. [22]

You can say a machine or a system is artificially intelligent when it is equipped with at least one and at most all intelligences in it.

II.2.11 What is Intelligence Composed of?

The intelligence is intangible. It is composed of [22]:

- Reasoning
- Problem Solving
- Linguistic Intelligence
- Learning
- Perception

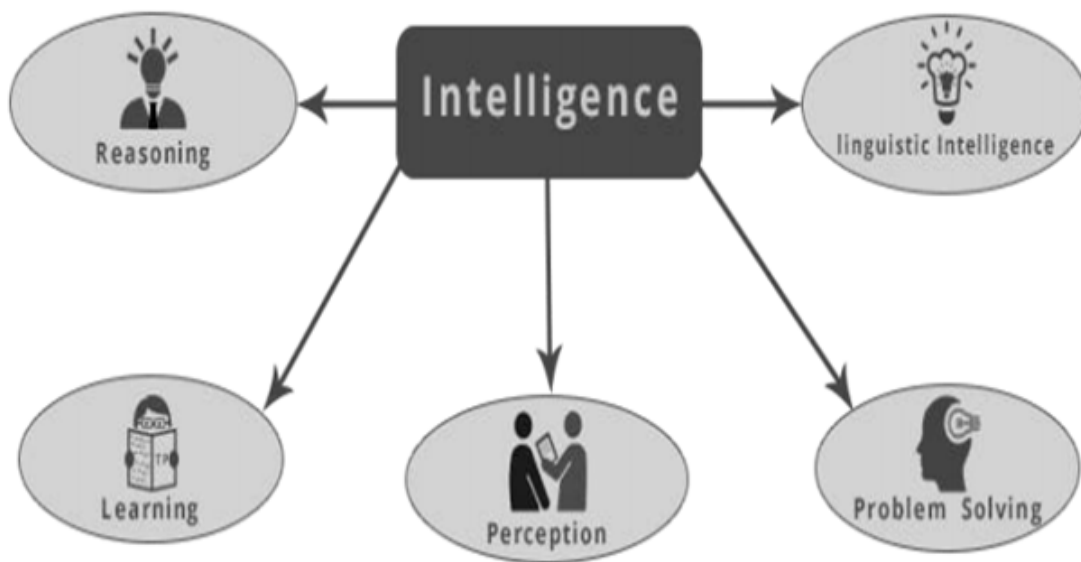


Figure II.2: Intelligence Composed. [22]

Let us go through all the components briefly:

1. **Reasoning:** It is the set of processes that enables us to provide basis for judgement, making decisions, and prediction.

There are broadly two types :

<i>Inductive Reasoning</i>	<i>Deductive Reasoning</i>
It conducts specific observations to makes broad general statements.	It starts with a general statement and examines the possibilities to reach a specific, logical conclusion.
Even if all of the premises are true in a statement, inductive reasoning allows for the conclusion to be false.	If something is true of a class of things in general, it is also true for all members of that class.
Example: "Nita is a teacher. All teachers are studious. Therefore, Nita is studious."	Example: "All women of age above 60 years are grandmothers. Shalini is 65 years. Therefore, Shalini is a grandmother."

Table II.4: Reasoning types. [22]

2. **Learning:** It is the activity of gaining knowledge or skill by studying, practising, being taught, or experiencing something. Learning enhances the awareness of the subjects of the study.

The ability of learning is possessed by humans, some animals, and AI-enabled systems. Learning is categorized as:

- **Auditory Learning** It is learning by listening and hearing. For example, students listening to recorded audio lectures.
- **Episodic Learning** To learn by remembering sequences of events that one has witnessed or experienced. This is linear and orderly.
- **Motor Learning** It is learning by precise movement of muscles. For example, picking objects, Writing, etc.
- **Observational Learning** To learn by watching and imitating others. For example, child tries to learn by mimicking her parent.
- **Perceptual Learning** It is learning to recognize stimuli that one has seen before. For example, identifying and classifying objects and situations.
- **Relational Learning** It involves learning to differentiate among various stimuli on the basis of relational properties, rather than absolute properties. For Example, adding 'little less' salt at the time of cooking potatoes that came up salty last time, when cooked with adding say a tablespoon of salt.
- **Spatial learning** It is learning through visual stimuli such as images, colors, maps, etc. For Example, a person can create roadmap in mind before actually following the road.
- **Stimulus-Response Learning** It is learning to perform a particular behavior when a certain stimulus is present. For example, a dog raises its ear on hearing doorbell.

3. **Problem solving:** It is the process in which one perceives and tries to arrive at a desired solution from a present situation by taking some path, which is blocked by known or unknown hurdles.

Problem solving also includes **decision making**, which is the process of selecting the best suitable alternative out of multiple alternatives to reach the desired goal are available.

4. **Perception:** It is the process of acquiring, interpreting, selecting, and organizing sensory information.

Perception presumes **sensing**. In humans, perception is aided by sensory organs. In the domain of AI, perception mechanism puts the data acquired by the sensors together in a meaningful manner.

5. **Linguistic Intelligence:** It is one's ability to use, comprehend, speak, and write the verbal and written language. It is important in interpersonal communication.[22]

II.2.12 Difference between Human and Machine Intelligence

- Humans perceive by patterns whereas the machines perceive by set of rules and data.
- Humans store and recall information by patterns, machines do it by searching algorithms. For example, the number 40404040 is easy to remember, store and recall as its pattern is simple.
- Humans can figure out the complete object even if some part of it is missing or distorted; whereas the machines cannot correctly.[22]

II.2.13 Real Life Applications of Research Areas

There is a large array of applications where AI is serving common people in their day-to-day lives[22]:




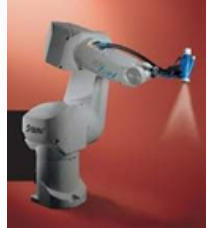

<i>Sr.No.</i>	<i>Research Areas</i>	<i>Real Life Application</i>
1	Expert Systems Examples – Flight-tracking systems, Clinical systems.	
2	Natural Language Processing Examples: Google Now feature, speech recognition, Automatic voice output.	
3	Neural Networks Examples – Pattern recognition systems such as face recognition, character recognition, handwriting recognition.	
4	Robotics Examples – Industrial robots for moving, spraying, painting, precision checking, drilling, cleaning, coating, carving, etc.	
5	Fuzzy Logic Systems Examples – Consumer electronics, automobiles, etc.	

Table II.5: Real Life Applications of Research Areas. [22]

II.2.14 Task Classification of AI

The domain of AI is classified into Formal tasks, Mundane tasks, and Expert tasks[22].(II.3)

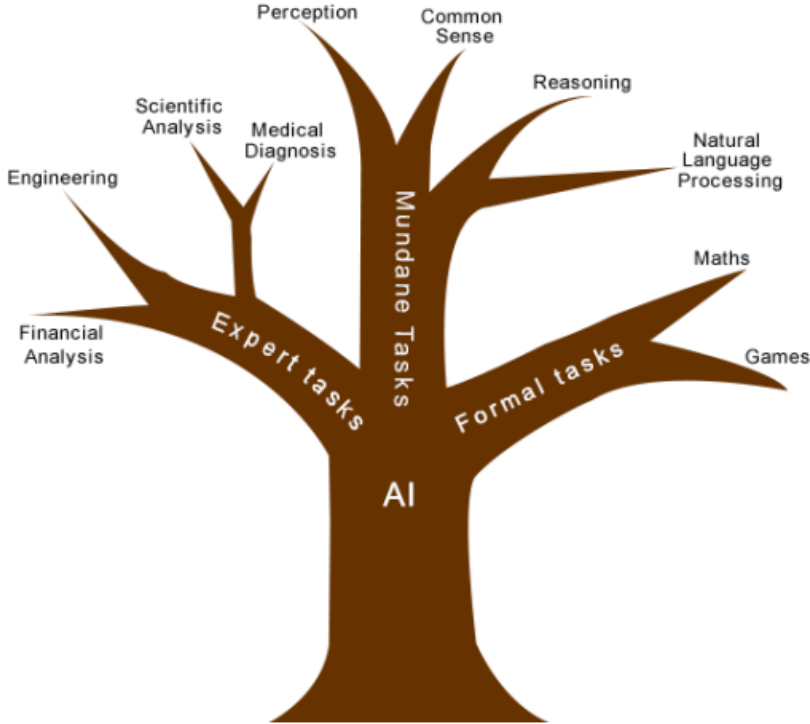


Figure II.3: Task Classification of AI. [22]

Humans learn **mundane (ordinary) tasks** since their birth. They learn by perception, speaking, using language, and locomotives. They learn Formal Tasks and Expert Tasks later, in that order.

For humans, the mundane tasks are easiest to learn. The same was considered true before trying to implement mundane tasks in machines. Earlier, all work of AI was concentrated in the mundane task domain.

Task Domains of Artificial Intelligence		
Mundane (Ordinary) Tasks	Formal Tasks	Expert Tasks
Perception: -Computer Vision -Speech, Voice	-Mathematics -Geometry -Logic -Integration and Differentiation	-Engineering -Fault Finding -Manufacturing -Monitoring
Natural Language Processing: -Understanding -Language Generation -Language Translation	Games: -Go -Chess (Deep Blue) -Ckeckers	Scientific Analysis
Common Sense	Verification	Financial Analysis
Reasoning	Theorem Proving	Medical Diagnosis
Planing		Creativity
Robotics: -Locomotive		

Table II.6: Table of Task Classificationof AI

Table II.7: Table of Task Classificationof AI. [22]

Later, it turned out that the machine requires more knowledge, complex knowledge representation, and complicated algorithms for handling mundane tasks. This is the reason **why AI work is more prospering in the Expert Task domain** now, as the expert task domain needs expert knowledge without common sense, which can be easier to represent and handle.[22]

II.3 Machine learning

Today's Artificial Intelligence (AI) has far surpassed the hype of blockchain and quantum computing. This is due to the fact that huge computing resources are easily available to the common man. The developers now take advantage of this in creating new Machine Learning models and to re-train the existing models for better performance and results. The easy availability of High Performance Computing (HPC) has resulted in a sudden increased demand for IT professionals having Machine Learning skills.

In this *section* , you will learn in detail about:

- What is the crux of machine learning?
- What are the different types in machine learning?
- What are the different algorithms available for developing machine learning models?
- What tools are available for developing these models?
- What are the programming language choices?
- What platforms support development and deployment of Machine Learning applications?
- What IDEs (Integrated Development Environment) are available?
- How to quickly upgrade your skills in this important area?[23].

II.3.1 Statistical Techniques

The development of today's AI applications started with using the age-old traditional statistical techniques. You must have used straight-line interpolation in schools to predict a future value. There are several other such statistical techniques which are successfully applied in developing so-called AI programs. We say "so-called" because the AI programs that we have today are much more complex and use techniques far beyond the statistical techniques used by the early AI programs.

Some of the examples of statistical techniques that are used for developing AI applications in those days and are still in practice are listed here:

- Regression
- Classification
- Clustering
- Probability Theories
- Decision Trees

Here we have listed only some primary techniques that are enough to get you started on AI without scaring you of the vastness that AI demands. If you are developing AI applications based on limited data, you would be using these statistical techniques.

However, today the data is abundant. To analyze the kind of huge data that we possess statistical techniques are of not much help as they have some limitations of their own. More advanced methods such as deep learning are hence developed to solve many complex problems.

As we move ahead , we will understand what Machine Learning is and how it is used for developing such complex AI applications.[23]

II.4 Machine Learning – Categories of Machine Learning

Machine Learning is broadly categorized under the following headings:(II.4)

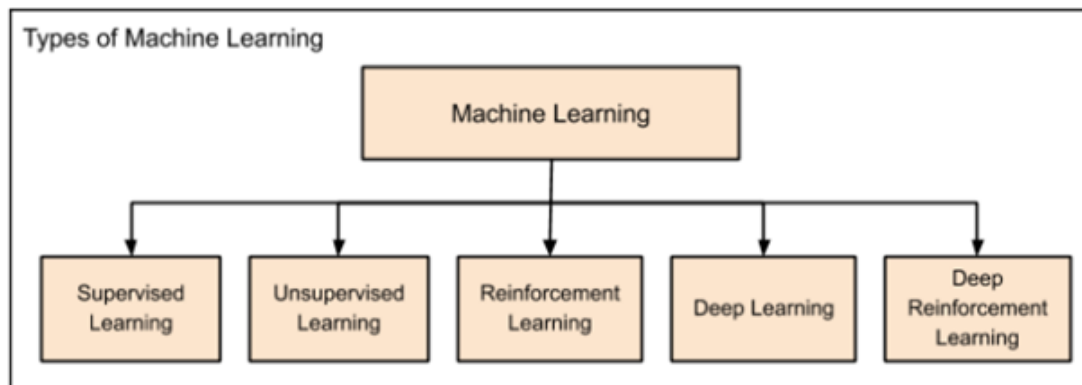


Figure II.4: Type of machine learning. [23]

Machine learning evolved from left to right as shown in the above diagram.

- Initially, researchers started out with Supervised Learning. This is the case of housing price prediction discussed earlier.
- This was followed by unsupervised learning, where the machine is made to learn on its own without any supervision.
- Scientists discovered further that it may be a good idea to reward the machine when it does the job the expected way and there came the Reinforcement Learning.
- Very soon, the data that is available these days has become so humongous that the conventional techniques developed so far failed to analyze the big data and provide us the predictions.
- Thus, came the deep learning where the human brain is simulated in the Artificial Neural Networks (ANN) created in our binary computers.
- The machine now learns on its own using the high computing power and huge memory resources that are available today.
- It is now observed that Deep Learning has solved many of the previously unsolvable problems.
- The technique is now further advanced by giving incentives to Deep Learning networks as awards and there finally comes Deep Reinforcement Learning.

Let us now study each of these categories in more detail.[23]

II.4.1 Supervised Learning

Supervised learning is analogous to training a child to walk. You will hold the child's hand, show him how to take his foot forward, walk yourself for a demonstration and so on, until the child learns to walk on his own.

II.4.1.1 Regression

Similarly, in the case of supervised learning, you give concrete known examples to the computer. You say that for given feature value x_1 the output is y_1 , for x_2 it is y_2 , for x_3 it is y_3 , and so on. Based on this data, you let the computer figure out an empirical relationship between x and y .

Once the machine is trained in this way with a sufficient number of data points, now you would ask the machine to predict Y for a given X . Assuming that you know the real value of Y for this given X , you will be able to deduce whether the machine's prediction is correct.

Thus, you will test whether the machine has learned by using the known test data. Once you are satisfied that the machine is able to do the predictions with a desired level of accuracy (say 80 to 90%) you can stop further training the machine.

Now, you can safely use the machine to do the predictions on unknown data points, or ask the machine to predict Y for a given X for which you do not know the real value of Y . This training comes under the regression that we talked about earlier.

II.4.1.2 Classification

You may also use machine learning techniques for classification problems. In classification problems, you classify objects of similar nature into a single group. For example, in a set of 100 students say, you may like to group them into three groups based on their heights - short, medium and long. Measuring the height of each student, you will place them in a proper group.

Now, when a new student comes in, you will put him in an appropriate group by measuring his height. By following the principles in regression training, you will train the machine to classify a student based on his feature – the height. When the machine learns how the groups are formed, it will be able to classify any unknown new student correctly. Once again, you would use the test data to verify that the machine has learned your technique of classification before putting the developed model in production.

Supervised Learning is where the AI really began its journey. This technique was applied successfully in several cases. You have used this model while doing the hand-written recognition on your machine. Several algorithms have been developed for supervised learning. You will learn about them in the following chapters.

II.4.2 Unsupervised Learning

In unsupervised learning, we do not specify a target variable to the machine, rather we ask machine “What can you tell me about X ?”. More specifically, we may ask questions such as given a huge data set X , “What are the five best groups we can make out of X ?” or “What features occur together most frequently in X ?”. To arrive at the answers to such questions, you can understand that the number of data points that the machine would require to deduce a strategy would be very large. In case of supervised learning, the machine can be trained with even about few thousands of data points. However, in case of unsupervised learning, the number of data points that is reasonably accepted for learning starts in a few millions. These days, the data is generally abundantly available. The data ideally requires curating. However, the amount of data that is continuously flowing in a social area network, in most cases data curation is an impossible task.

The following figure shows the boundary between the yellow and red dots as determined by unsupervised machine learning. You can see it clearly that the machine would be able to deter-

mine the class of each of the black dots with a fairly good accuracy.(II.5)

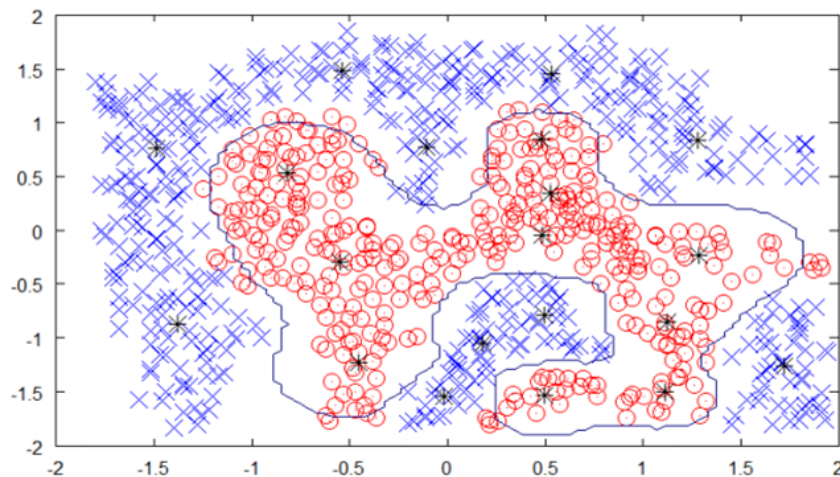


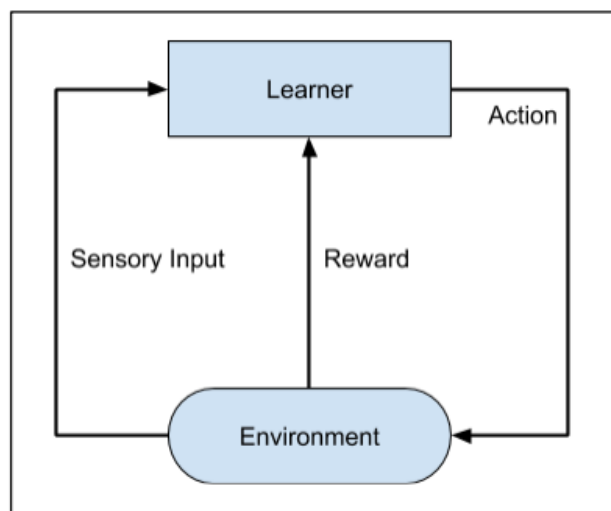
Figure II.5: Unsupervised Learning. [23]

The unsupervised learning has shown a great success in many modern AI applications, such as face detection, object detection, and so on.

II.4.3 Reinforcement Learning

Consider training a pet dog, we train our pet to bring a ball to us. We throw the ball at a certain distance and ask the dog to fetch it back to us. Every time the dog does this right, we reward the dog. Slowly, the dog learns that doing the job rightly gives him a reward and then the dog starts doing the job right way every time in future. Exactly, this concept is applied in “Reinforcement” type of learning. The technique was initially developed for machines to play games. The machine is given an algorithm to analyze all possible moves at each stage of the game. The machine may select one of the moves at random. If the move is right, the machine is rewarded, otherwise it may be penalized. Slowly, the machine will start differentiating between right and wrong moves and after several iterations would learn to solve the game puzzle with a better accuracy. The accuracy of winning the game would improve as the machine plays more and more games.

The entire process may be depicted in the following diagram:



This technique of machine learning differs from the supervised learning in that you need not

supply the labelled input/output pairs. The focus is on finding the balance between exploring the new solutions versus exploiting the learned solutions.

II.4.4 Deep Learning

The deep learning is a model based on Artificial Neural Networks (ANN), more specifically Convolutional Neural Networks (CNN)s. There are several architectures used in deep learning such as deep neural networks, deep belief networks, recurrent neural networks, and convolutional neural networks.

These networks have been successfully applied in solving the problems of computer vision, speech recognition, natural language processing, bioinformatics, drug design, medical image analysis, and games. There are several other fields in which deep learning is proactively applied. The deep learning requires huge processing power and humongous data, which is generally easily available these days.

We will talk about deep learning more in detail in the coming chapters.

II.4.5 Deep Reinforcement Learning

The Deep Reinforcement Learning (DRL) combines the techniques of both deep and reinforcement learning. The reinforcement learning algorithms like Q-learning are now combined with deep learning to create a powerful DRL model. The technique has been with a great success in the fields of robotics, video games, finance and healthcare. Many previously unsolvable problems are now solved by creating DRL models. There is lots of research going on in this area and this is very actively pursued by the industries.

So far, you have got a brief introduction to various machine learning models, now let us explore slightly deeper into various algorithms that are available under these models.

II.5 Machine Learning – Supervised Learning

Supervised learning is one of the important models of learning involved in training machines. This chapter talks in detail about the same.

II.5.1 Algorithms for Supervised Learning

There are several algorithms available for supervised learning. Some of the widely used algorithms of supervised learning are as shown below:

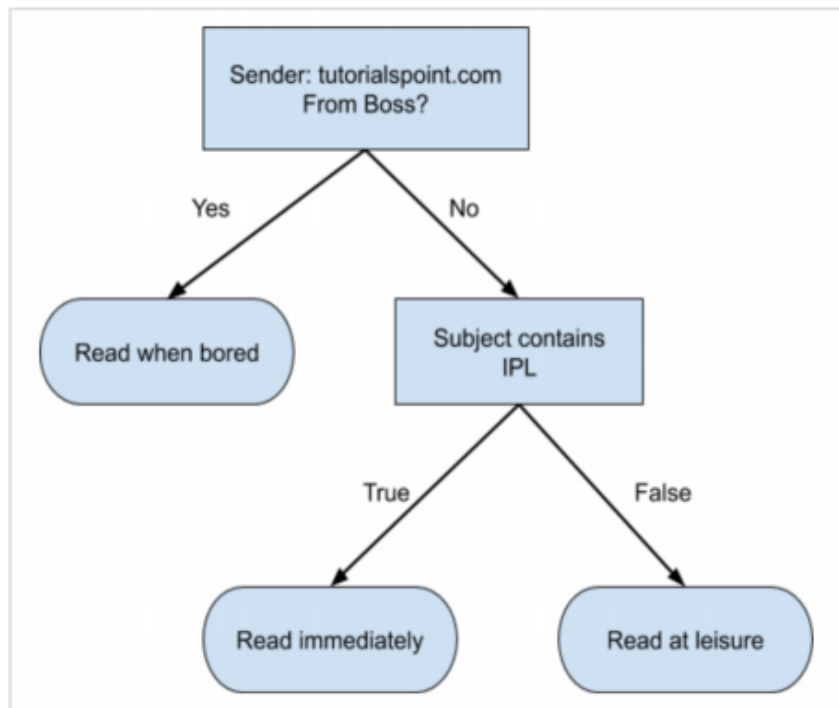
- k-Nearest Neighbors
- Decision Trees
- Naive Bays
- Logistic Regression
- Support Vector Machines

II.5.2 k-Nearest Neighbours

The k-Nearest Neighbours, which is simply called kNN is a statistical technique that can be used for solving for classification and regression problems.

II.5.3 Decision Trees

A simple decision tree in a flowchart format is shown below:



You would write a code to classify your input data based on this flowchart. The flowchart is self-explanatory and trivial. In this scenario, you are trying to classify an incoming email to decide when to read it.

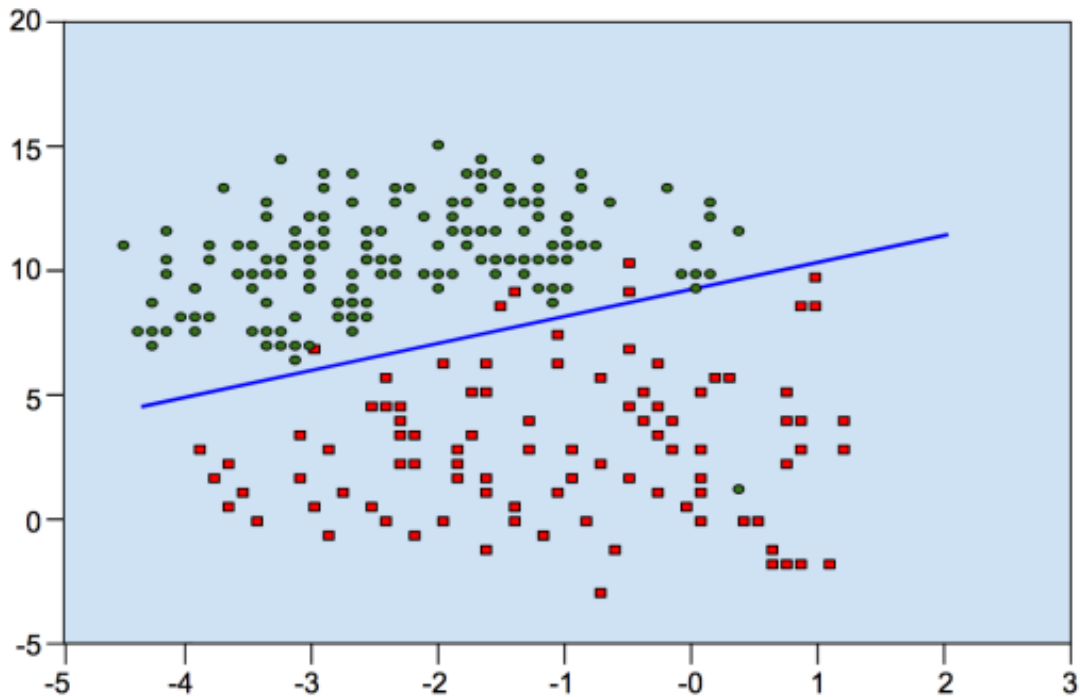
In reality, the decision trees can be large and complex. There are several algorithms available to create and traverse these trees. As a Machine Learning enthusiast, you need to understand and master these techniques of creating and traversing decision trees.

II.5.4 Naive Bayes

Naive Bayes is used for creating classifiers. Suppose you want to sort out (classify) fruits of different kinds from a fruit basket. You may use features such as color, size and shape of a fruit, For example, any fruit that is red in color, is round in shape and is about 10 cm in diameter may be considered as Apple. So to train the model, you would use these features and test the probability that a given feature matches the desired constraints. The probabilities of different features are then combined to arrive at a probability that a given fruit is an Apple. Naive Bayes generally requires a small number of training data for classification.

II.5.5 Logistic Regression

Look at the following diagram. It shows the distribution of data points in XY plane.[23]



From the diagram, we can visually inspect the separation of red dots from green dots. You may draw a boundary line to separate out these dots. Now, to classify a new data point, you will just need to determine on which side of the line the point lies.

II.5.6 Machine Learning – Scikit-learn Algorithm

Fortunately, most of the time you do not have to code the algorithms mentioned in the previous lesson. There are many standard libraries which provide the ready-to-use implementation of these algorithms. One such toolkit that is popularly used is scikit-learn.

The figure below illustrates the kind of algorithms which are available for your use in this library.(II.6)

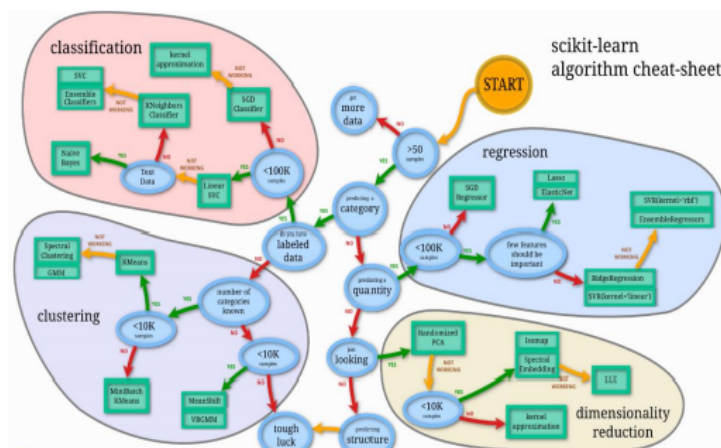


Figure II.6: Scikit-learn Algorithm cheat-sheet.

The use of these algorithms is trivial and since these are well and field tested, you can safely use them in your AI applications. Most of these libraries are free to use even for commercial purposes.

II.6 Machine Learning – Unsupervised Learning

So far what you have seen is making the machine learn to find out the solution to our target. In regression, we train the machine to predict a future value. In classification, we train the machine to classify an unknown object in one of the categories defined by us. In short, we have been training machines so that it can predict Y for our data X. Given a huge data set and not estimating the categories, it would be difficult for us to train the machine using supervised learning. What if the machine can look up and analyze the big data running into several Gigabytes and Terabytes and tell us that this data contains so many distinct categories?

As an example, consider the voter's data. By considering some inputs from each voter (these are called features in AI terminology), let the machine predict that there are so many voters who would vote for X political party and so many would vote for Y, and so on. Thus, in general, we are asking the machine given a huge set of data points X, "What can you tell me about X?". Or it may be a question like "What are the five best groups we can make out of X?". Or it could be even like "What three features occur together most frequently in X?".

This is exactly the Unsupervised Learning is all about.

II.6.1 Algorithms for Unsupervised Learning

Let us now discuss one of the widely used algorithms for classification in unsupervised machine learning.

II.6.2 k-means clustering

Clustering is a type of unsupervised learning that automatically forms clusters of similar things. It is like automatic classification. You can cluster almost anything, and the more similar the items are in the cluster, the better the clusters are. In this chapter, we are going to study one type of clustering algorithm called k-means. It is called k-means because it finds 'k' unique clusters, and the center of each cluster is the mean of the values in that cluster.

II.6.3 Cluster Identification

Cluster identification tells an algorithm, "Here's some data. Now group similar things together and tell me about those groups." The key difference from classification is that in classification you know what you are looking for. While that is not the case in clustering.

Clustering is sometimes called unsupervised classification because it produces the same result as classification does but without having predefined classes.

Now, we are comfortable with both supervised and unsupervised learning. To understand the rest of the machine learning categories, we must first understand Artificial Neural Networks (ANN), which we will learn in the next chapter...

II.6.4 Machine Learning – Artificial Neural Networks

The idea of artificial neural networks was derived from the neural networks in the human brain. The human brain is really complex. Carefully studying the brain, the scientists and engineers

came up with an architecture that could fit in our digital world of binary computers. One such typical architecture is shown in the diagram below:

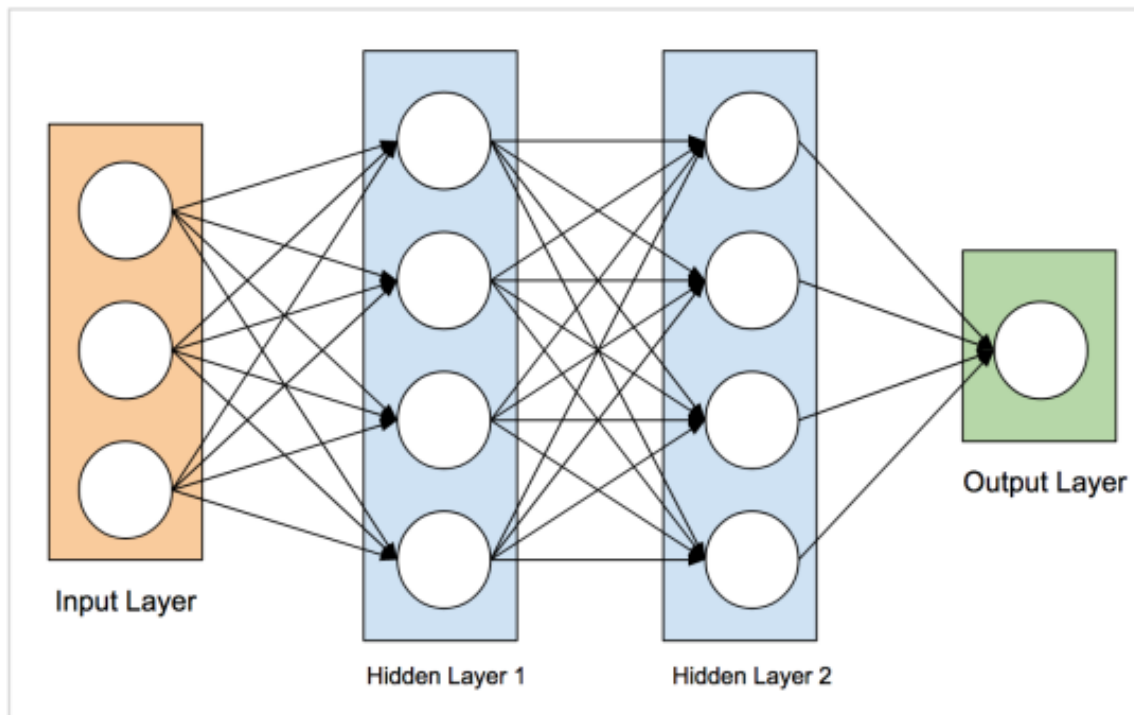


Figure II.7: architecture neural network. [23]

There is an input layer which has many sensors to collect data from the outside world. On the right hand side, we have an output layer that gives us the result predicted by the network. In between these two, several layers are hidden. Each additional layer adds further complexity in training the network, but would provide better results in most of the situations. There are several types of architectures designed which we will discuss now.

II.6.5 ANN Architectures

The diagram below shows several ANN architectures developed over a period of time and are in practice today.(II.8) [24]

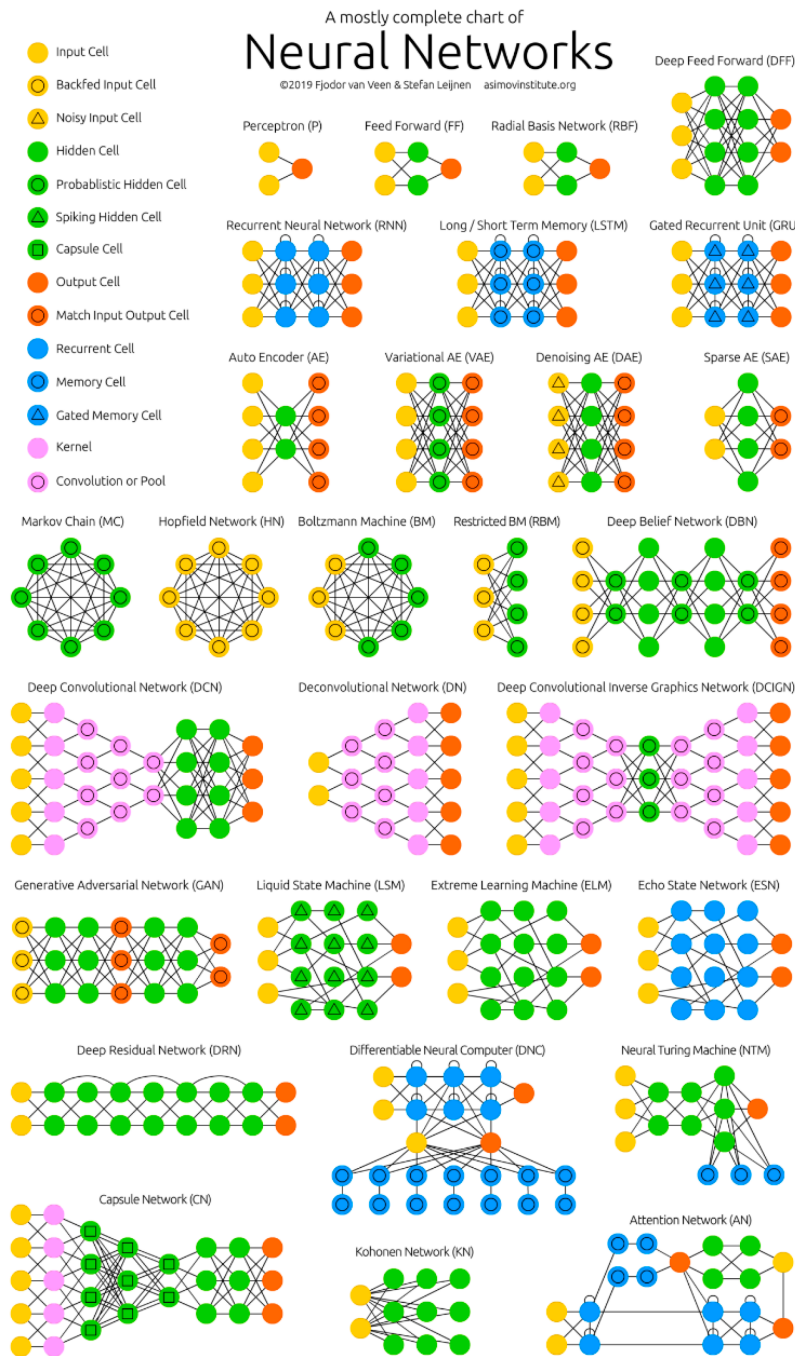


Figure II.8: A mostly complet chart of Neural Networks. [24]

Each architecture is developed for a specific type of application. Thus, when you use a neural network for your machine learning application, you will have to use either one of the existing architecture or design your own. The type of application that you finally decide upon depends on your application needs. There is no single guideline that tells you to use a specific network architecture.

II.7 Machine Learning – Deep Learning

Deep Learning uses ANN. First we will look at a few deep learning applications that will give you an idea of its power.

II.7.1 Applications

Deep Learning has shown a lot of success in several areas of machine learning applications.

Self-driving Cars: The autonomous self-driving cars use deep learning techniques. They generally adapt to the ever changing traffic situations and get better and better at driving over a period of time.

Speech Recognition: Another interesting application of Deep Learning is speech recognition. All of us use several mobile apps today that are capable of recognizing our speech. Apple's Siri, Amazon's Alexa, Microsoft's Cortana and Google's Assistant – all these use deep learning techniques.

Mobile Apps: We use several web-based and mobile apps for organizing our photos. Face detection, face ID, face tagging, identifying objects in an image – all these use deep learning.

II.7.2 Untapped Opportunities of Deep Learning

After looking at the great success deep learning applications have achieved in many domains, people started exploring other domains where machine learning was not so far applied. There are several domains in which deep learning techniques are successfully applied and there are many other domains which can be exploited. Some of these are discussed here:

- Agriculture is one such industry where people can apply deep learning techniques to improve the crop yield.
- Consumer finance is another area where machine learning can greatly help in providing early detection on frauds and analyzing customer's ability to pay.
- Deep learning techniques are also applied to the field of medicine to create new drugs and provide a personalized prescription to a patient. The possibilities are endless and one has to keep watching as the new ideas and developments pop up frequently.

II.7.3 What is Required for Achieving More Using Deep Learning?

To use deep learning, supercomputing power is a mandatory requirement. You need both memory as well as the CPU to develop deep learning models. Fortunately, today we have an easy availability of HPC – High Performance Computing. Due to this, the development of the deep learning applications that we mentioned above became a reality today and in the future too we can see the applications in those untapped areas that we discussed earlier.

Now, we will look at some of the limitations of deep learning that we must consider before using it in our machine learning application.

II.7.4 Deep Learning -Disadvantages

Some of the important points that you need to consider before using deep learning are listed below: [23]

- Black Box approach

- Duration of Development
- Amount of Data
- Computationally Expensive

We will now study each one of these limitations in detail.

II.7.5 Black Box approach

An ANN is like a blackbox. You give it a certain input and it will provide you a specific output. The following diagram shows you one such application where you feed an animal image to a neural network and it tells you that the image is of a dog.(II.9)

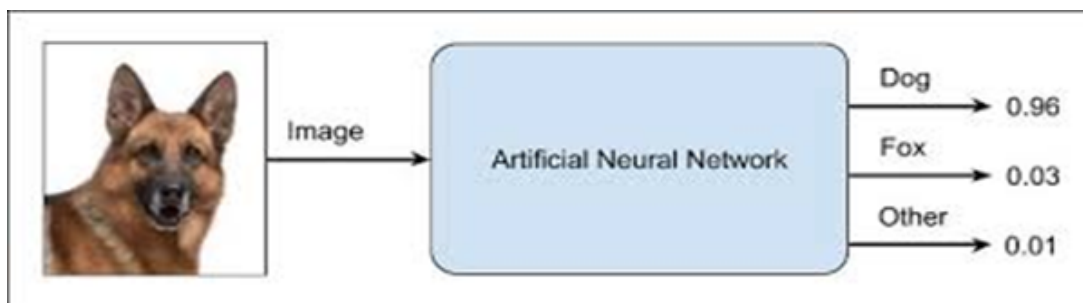
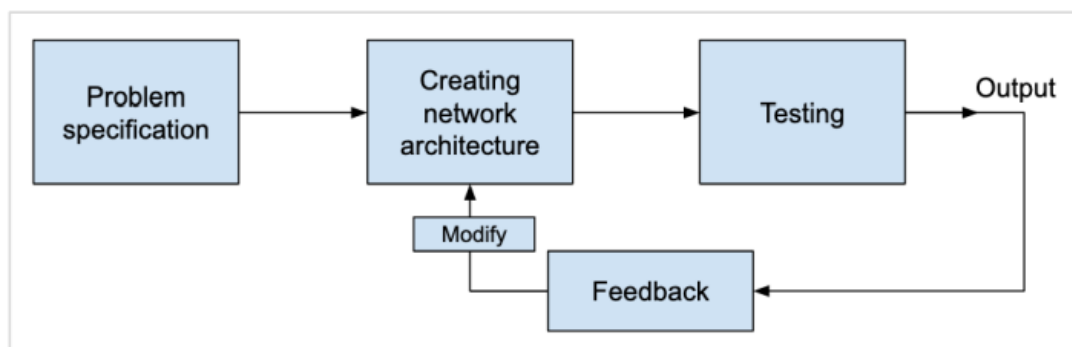


Figure II.9: neural network and it tells you that the image is of a dog. [23]

Why this is called a black-box approach is that you do not know why the network came up with a certain result. You do not know how the network concluded that it is a dog? Now consider a banking application where the bank wants to decide the creditworthiness of a client. The network will definitely provide you an answer to this question. However, will you be able to justify it to a client? Banks need to explain it to their customers why the loan is not sanctioned?

II.7.6 Duration of Development

The process of training a neural network is depicted in the diagram below:



You first define the problem that you want to solve, create a specification for it, decide on the input features, design a network, deploy it and test the output. If the output is not as expected, take this as a feedback to restructure your network. This is an iterative process and may require several iterations until the time network is fully trained to produce desired outputs.

II.7.7 Amount of Data

The deep learning networks usually require a huge amount of data for training, while the traditional machine learning algorithms can be used with a great success even with just a few thousands of data points. Fortunately, the data abundance is growing at 40% per year and CPU processing power is growing at 20% per year.

II.7.8 Computationally Expensive

Training a neural network requires several times more computational power than the one required in running traditional algorithms. Successful training of deep Neural Networks may require several weeks of training time.

In contrast to this, traditional machine learning algorithms take only a few minutes/hours to train. Also, the amount of computational power needed for training deep neural network heavily depends on the size of your data and how deep and complex the network is?

After having an overview of what Machine Learning is, its capabilities, limitations, and applications, let us now dive into learning “Machine Learning”.

II.7.9 Mathematical Notation

Most of the machine learning algorithms are heavily based on mathematics. The level of mathematics that you need to know is probably just a beginner level. What is important is that you should be able to read the notation that mathematicians use in their equations.

For example - if you are able to read the notation and comprehend what it means, you are ready for learning machine learning. If not, you may need to brush up your mathematics knowledge.

II.7.10 Probability Theory

Here is an example to test your current knowledge of probability theory: Classifying with conditional probabilities.

II.7.11 Visualization

In many cases, you will need to understand the various types of visualization plots to understand your data distribution and interpret the results of the algorithm’s output.

II.8 Machine Learning – Implementing Machine Learning

To develop ML applications, you will have to decide on the platform, the IDE and the language for development. There are several choices available. Most of these would meet your requirements easily as all of them provide the implementation of AI algorithms discussed so far.

If you are developing the ML algorithm on your own, the following aspects need to be understood carefully:

The language of your choice – this essentially is your proficiency in one of the languages supported in ML development.

The IDE that you use – This would depend on your familiarity with the existing IDEs and your comfort level.

Development platform – There are several platforms available for development and deployment.

Most of these are free-to-use. In some cases, you may have to incur a license fee beyond a certain amount of usage. Here is a brief list of choice of languages, IDEs and platforms for your ready reference.

II.8.1 Language Choice

Here is a list of languages that support ML development:

- Python
- R
- Matlab
- Octave
- Julia
- C++
- C

This list is not essentially comprehensive; however, it covers many popular languages used in machine learning development. Depending upon your comfort level, select a language for the development, develop your models and test.

II.8.2 IDEs

Here is a list of IDEs which support ML development:

- R Studio
- Pycharm
- iPython/Jupyter Notebook
- Julia
- Spyder
- Anaconda
- Google –Colab
- Rodeo

The above list is not essentially comprehensive. Each one has its own merits and demerits. The reader is encouraged to try out these different IDEs before narrowing down to a single one.

II.8.3 Platforms

Here is a list of platforms on which ML applications can be deployed:

- IBM
- Microsoft Azure
- Google Cloud

- Amazon
- Mlflow

Once again this list is not exhaustive. The reader is encouraged to sign-up for the abovementioned services and try them out themselves.

II.9 Conclusion

This chapter has introduced you to Machine Learning. Now, you know that Machine Learning is a technique of training machines to perform the activities a human brain can do, albeit bit faster and better than an average human-being. Today we have seen that the machines can beat human champions in games such as Chess, AlphaGO, which are considered very complex. You have seen that machines can be trained to perform human activities in several areas and can aid humans in living better lives. [24]

Chapter III

Deep Learning on Graphs

III.1 INTRODUCTION

Over the past decade, deep learning has become the "crown jewel" of artificial intelligence and machine learning as it has shown superior performance in several areas. The expressive power of deep learning for extracting complex patterns from underlying data is well recognized. On the other hand, graphs are ubiquitous in the real world, representing objects and their relationships in diverse domains. Graphs are also known to have complex structures that can contain rich base values. As a result, how deep learning methods are used to analyze graph data has attracted much research interest. This problem is not trivial because there are many challenges in applying traditional deep learning architectures to graphs

- **Irregular structures of graphs.**
- **Heterogeneity and diversity of graphs.**
- **Large-scale graphs.**
- **Incorporating interdisciplinary knowledge.**

In this chapter, we try reviewing deep learning methods on graphs. Specifically, as shown in Figure (III.1), we divide the existing methods into five categories based on their model architectures and training strategies.

We summarize some of the main characteristics of these categories in Table III.1 based on the high-level distinctions.[25]

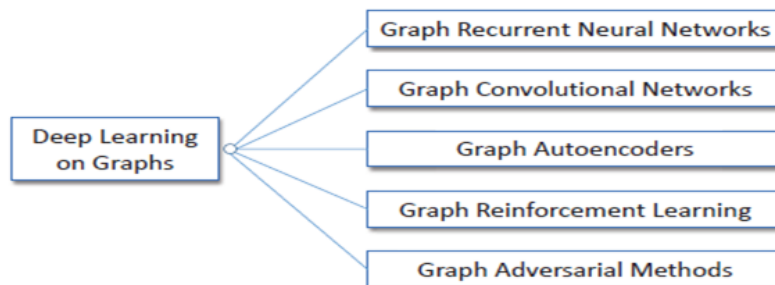


Figure III.1: A categorization of deep learning methods on graphs. [25]

<i>Category</i>	<i>Basic Assumptions/Aims</i>	<i>Main Functions</i>
Graph convolutional networks	Common local and global structural patterns of graphs	Definitions of states for nodes or graphs
Graph convolutional networks	Common local and global structural patterns of graphs	Graph convolution and readout operations
Graph autoencoders	Low-rank structures of graphs	Unsupervised node representation learning
Graph reinforcement learning	Feedbacks and constraints of graph tasks	Graph-based actions and rewards
Graph adversarial methods	The generalization ability and robustness of graph-based models	Graph adversarial trainings and attacks

Table III.1: Main Distinctions among Deep Learning Methods on Graphs. [25]

In the following sections, we provide a comprehensive and detailed overview of these methods, mainly by following their development history and the various ways these methods solve the challenges posed by graphs. We also analyze the differences between these models and delve into how to composite different architectures. Finally, we briefly outline the applications of these models, introduce several open libraries.

III.2 NOTATIONS AND PRELIMINARIES

$G = (V, E)$	A graph
N, M	The number of nodes and edges
$V = (v^1, \dots, v^N)$	The set of nodes
$\mathbf{F}^V, \mathbf{F}^E$	The attributes/features of nodes and edges
\mathbf{A}	The adjacency matrix
$\mathbf{D}(i, i) = \sum_j \mathbf{A}(i, j)$	The diagonal degree matrix
$\mathbf{Q} \wedge \mathbf{Q}^T = \mathbf{L}$	The Laplacian matrix
$\mathbf{L} = \mathbf{D} - \mathbf{A}$	The eigendecomposition of \mathbf{L}
$\mathbf{P} = \mathbf{D}^{-1} - \mathbf{A}$	The transition matrix
$N_k(i), N(i)$	The k-step and 1-step neighbors of v_i
\mathbf{H}^l	The hidden representation in the l^{th} layer
f_l	The dimensionality of \mathbf{H}^l
$\rho(\cdot)$	Some non-linear activation function
$\mathbf{X}_1 \odot \mathbf{X}_2$	The element-wise multiplication
Θ	Learnable parameters
s	The sample size

Table III.2: A Table for Commonly Used Notations. [25]

III.3 GRAPH RECURRENT NEURAL NETWORKS

Recurrent neural networks (RNNs) is a fact standard in modeling sequential data. In this section, we review Graph RNNs which can capture recursive and sequential patterns of graphs. Graph RNNs can be broadly divided into two categories: node-level RNNs and graph-level RNNs. The main distinction lies in whether the patterns lie at the node-level and are modeled by node states, or at the graph-level and are modeled by a common graph state.

III.3.1 Node-level RNNs

Node-level RNNs for graphs, which are also referred to as graph neural networks (GNNs)³, can be dated back to the "pre-deep learning" era. The idea behind a GNN is simple: to encode graph structural information, each node is represented by a low-dimensional state vector. Motivated by recursive neural networks.

For graph-focused tasks, the authors suggested adding a special node with unique attributes to represent the entire graph. To learn the model parameters, semi-supervised method is adopted.

GNN plays two important roles. In retrospect, a GNN unifies some of the early methods used

for processing graph data, such as recursive neural networks and Markov chains. Looking toward the future, the general idea underlying GNNs has profound inspirations. In fact, GNNs and GCNs can be unified into some common frameworks, and a GNN is equivalent to a GCN that uses identical layers to reach stable states. Although they are conceptually important, GNNs have several drawbacks.

Intuitively, a “contraction map” requires that the distance between any two points can only “contract”, which severely limits the modeling ability. Second, because many iterations are needed to reach a stable state between gradient descend steps, GNNs are computationally expensive. Because of these drawbacks and perhaps a lack of computational power and lack of research interests, GNNs did not become a focus of general research.

A notable improvement to GNNs is gated graph sequence neural networks (GGS-NNs) with the modifications. Most importantly, the authors replaced the recursive definition with a GRU, thus removing the “contraction map” requirement and supporting modern optimization techniques.

The authors proposed using several networks operating in sequence to produce sequence outputs and showed that their method could be applied to sequence-based tasks such as program verification. SSE took a similar approach. However, instead of using a GRU in the calculation, SSE adopted stochastic fixed point gradient descent to accelerate the training process. This scheme basically alternates between calculating steady node states using local neighborhoods and optimizing the model parameters, with both calculations in stochastic mini-batches.

<i>Category</i>	<i>Method</i>	<i>Recursive/sequential patterns of graphs</i>	<i>Time Complexity</i>	<i>Other Improvements</i>
Node-level	GNN	A recursive definition of node states	$O(MI_f)$	-
	GGS-NNs		$O(MT)$	Sequence outputs
	SSE		$O(d_{avg}S)$	-
Graph-level	You et al.	Generate nodes and edges in an autoregressive manner	$O(N^2)$	-
	DGNN	Capture the time dynamics of the formation of nodes and edges	$O(Md_{avg})$	-
	RMGCNN	Recursively reconstruct the graph	$O(M)$ or $O(MN)$	GCN layers
	Dynamic GCN	Gather node representations in different time slices	$O(Mt)$	GCN layers

Table III.3: The Main Characteristics of Graph Recurrent Neural Network (Graph RNNs). [25]

III.3.2 Graph-level RNNs

In this subsection, we review how to apply RNNs to capture graph-level patterns, e.g., temporal patterns of dynamic graphs or sequential patterns at different levels of graph granularities. In graph-level RNNs, instead of applying one RNN to each node to learn the node states, a single RNN is applied to the entire graph to encode the graph states.

You et al. applied Graph RNNs to the graph generation problem. they adopted two RNNs: one to generate new nodes and the other to generate edges for the newly added node in an autoregressive manner. They showed that such hierarchical RNN architectures learn more effectively from input graphs than do the traditional rule-based graph generative models while having a reasonable time complexity.

To capture the temporal information of dynamic graphs, dynamic graph neural network (DGNN) was proposed that used a time-aware LSTM to learn node representations. When a new edge is established, DGNN used the LSTM to update the representation of the two interacting nodes as well as their immediate neighbors, i.e., considering the one-step propagation

effect. The authors showed that the time-aware LSTM could model the establishing orders and time intervals of edge formations well, which in turn benefited a range of graph applications.

III.4 GRAPH CONVOLUTIONAL NETWORKS

Graph convolutional networks (GCNs) are the hottest topic in graph-based deep learning. modern GCNs learn the common local and global structural patterns of graphs through designed convolution and readout functions. Because most GCNs can be trained with task-specific loss via backpropagation, we focus on the adopted architectures. We first discuss the convolution operations, then move to the readout operations and some other improvements. We summarize the main characteristics of GCNs surveyed in this paper in Table 3.(III.4)

III.4.1 Convolution Operations

Graph convolutions can be divided into two groups: spectral convolutions, which perform convolution by transforming node representations into the spectral domain using the graph Fourier transform or its extensions, and spatial convolutions, which perform convolution by considering node neighborhoods. Note that these two groups can overlap, for example, when using a polynomial spectral kernel.

III.4.1.1 Spectral Methods

Convolution is the most fundamental operation in CNNs. However, the standard convolution operation used for images or text cannot be directly applied to graphs because graphs lack a grid structure. Bruna et al first introduced convolution for graph data from the spectral domain using the graph Laplacian matrix L , which plays a similar role as the Fourier basis in signal processing.

III.4.1.2 The Efficiency Aspect

The recursive definition. From this aspect, a GNN can be regarded as a GCN with a large number of identical layers to reach stable states, i.e., a GNN uses a fixed function with fixed parameters to iteratively update the node hidden states until reaching an equilibrium, while a GCN has a preset number of layers and each layer contains different parameters.

III.4.1.3 The Aspect of Multiple Graphs

A parallel series of works has focuses on generalizing graph convolutions to multiple graphs of arbitrary sizes. Neural FPs proposed a spatial method that also used the first-order neighbors.

<i>Method</i>	<i>Type</i>	<i>Convolution</i>	<i>Readout</i>	<i>T.C.</i>	<i>M.G.</i>	<i>Other Characteristics</i>
Bruna et al.	Spectral	Interpolation Kernel	Hierarchical Clustering + FC	$O(N^3)$	No	-
Henaff et al.	Spectral	Interpolation Kernel	Hierarchical Clustering + FC	$O(N^3)$	No	Constructing the Graph
ChebNet	Spectral/Spatial	Polynomial	Hierarchical Clustering	$O(M)$	Yes	-
Kipf&Welling	Spectral/Spatial	First-order	-	$O(M)$	-	-
CayletNet	Spectral	Polynomial	-	$O(M)$	No	-
GWNN	Spectral	Wavelet Transform	-	$O(M)$	No	-
Neural FPs	Spectral	First-order	Sum	$O(M)$	Yes	-
PATCHY-SAN	Spectral	Polynomial + Order	Order + Pooling	$O(M \log N)$	Yes	An Order for Neighbors
LGCN	Spectral	First-order + Order	-	$O(M)$	Yes	An Order for Neighbors
SortPooling	Spectral	First-order	Order + Pooling	$O(M)$	Yes	An Order for Nodes
DCNN	Spectral	Polynomial Diffusion	Mean	$O(N^2)$	Yes	Edge Features
DGCN	Spectral	First-order + Diffusion	-	$O(N^2)$	-	-
MPNNs	Spectral	First-order	Set2set	$O(M)$	Yes	General Framework
GraphSAGE	Spectral	First-order + Sampling	-	$O(M \log N)$	-	General Framework
MoNet	Spectral	First-order	Hierarchical Clustering	$O(M)$	Yes	General Framework
GNs	Spectral	First-order	Whole Graph Representation	$O(M)$	Yes	General Framework
Kearnes et al	Spectral	Weave module	Fuzzy Histogram	$O(N^2)$	Yes	Edge Features
DiffPool	Spectral	Various	Hierarchical Clustering	$O(N^2)$	Yes	Differentiable Pooling
GAT	Spectral	First-order	-	$O(M)$	Yes	Attention
GaAN	Spectral	First-order	-	$O(NsL)$	Yes	Attention
HAN	Spectral	Meta-path Neighbors	-	$O(M\phi)$	Yes	Attention
CLN	Spectral	First-order	-	$O(M)$	-	-
PPNP	Spectral	First-order	-	$O(M)$	-	Teleportation Connection
JK-Nets	Spectral	Various	-	$O(M)$	Yes	Jumping Connection
ECC	Spectral	First-order	Hierarchical Clustering	$O(M)$	Yes	Edge Features
R-GCNs	Spectral	First-order	-	$O(M)$	-	Edge Features
LGNN	Spectral	First-order + LINE graph	-	$O(M)$	-	Edge Features
PinSage	Spectral	Random Walk	-	$O(NsL)$	-	Neighborhood Sampling
StochasticGCN	Spectral	First-order + Sampling	-	$O(NsL)$	-	Neighborhood Sampling
FastGCN	Spectral	First-order + Sampling	-	$O(NsL)$	Yes	Layer-wise Sampling
Adapt	Spectral	First-order + Sampling	-	$O(NsL)$	Yes	Layer-wise Sampling
Li et al.	Spectral	First-order	-	$O(M)$	-	Theoretical analysis
SGC	Spectral	Polynomial	-	$O(M)$	Yes	Theoretical analysis
GFNN	Spectral	Polynomial	-	$O(M)$	Yes	Theoretical analysis
GIN	Spectral	First-order	Sum + MLP	$O(M)$	Yes	Theoretical analysis
DGI	Spectral	First-order	-	$O(M)$	Yes	Unsupervised training

Table III.4: A Comparison among Different Graph Convolutional Networks (GCNs). T.C. = Time Complexity, M.G. = Multiple Graphs. [25]

III.4.1.4 Frameworks

MPNNs were proposed as a unified framework for the graph convolution operation in the spatial domain using message-passing functions.

III.4.2 Readout Operations

Using graph convolution operations, useful node features can be learned to solve many node-focused tasks. However, to tackle graph-focused tasks, node information needs to be aggregated to form a graph-level representation. Based on a regular and local neighborhood, standard CNNs conduct multiple stride convolutions or poolings to gradually reduce the resolution. Since graphs lack a grid structure, these existing methods cannot be used directly.

III.4.2.1 Statistics

The most basic order-invariant operations involve simple statistics such as summation, averaging or max-pooling.

III.4.2.2 Hierarchical Clustering

graphs are known to exhibit rich hierarchical structures, which can be explored by hierarchical clustering methods. However, hierarchical clustering methods are all independent of the graph convolutions. To solve that problem, DiffPool proposed a differentiable hierarchical clustering algorithm jointly trained with the graph convolutions. Specifically, the authors proposed learning a soft cluster assignment matrix in each layer using the hidden representations.

III.4.2.3 Imposing Orders and Others

PATCHY-SAN and SortPooling took the idea of imposing a node order and then resorted to standard 1-D pooling as in CNNs. Whether these methods can preserve order invariance depends on how the order is imposed, which is another research field. However, whether imposing a node order is a natural choice for graphs and if so, what the best node orders are constituting still on-going research topics.

Heuristics. In GNNs, the authors suggested adding a special node connected to all nodes to represent the entire graph. Similarly, GNs proposed to directly learn the representation of the entire graph by receiving messages from all nodes and edges.

III.4.2.4 Summary

In short, statistics such as averaging or summation are the simplest readout operations, while hierarchical clustering algorithms jointly trained with graph convolutions are more advanced but are also more sophisticated. Other methods such as adding a pseudo node or imposing a node order have also been investigated.[25]

III.4.3 Improvements and Discussions

Many techniques have been introduced to further improve GCNs. Note that some of these methods are general and could be applied to other deep learning models on graphs as well.

III.4.3.1 Attention Mechanism

In the aforementioned GCNs, the node neighborhoods are aggregated with equal or pre-defined weights. However, the influences of neighbors can vary greatly; thus, they should be learned during training rather than being predetermined. Inspired by the attention mechanism, graph attention network (GAT) introduces the attention mechanism into GCNs by modifying the convolution operation.

III.4.3.2 Residual and Jumping Connections

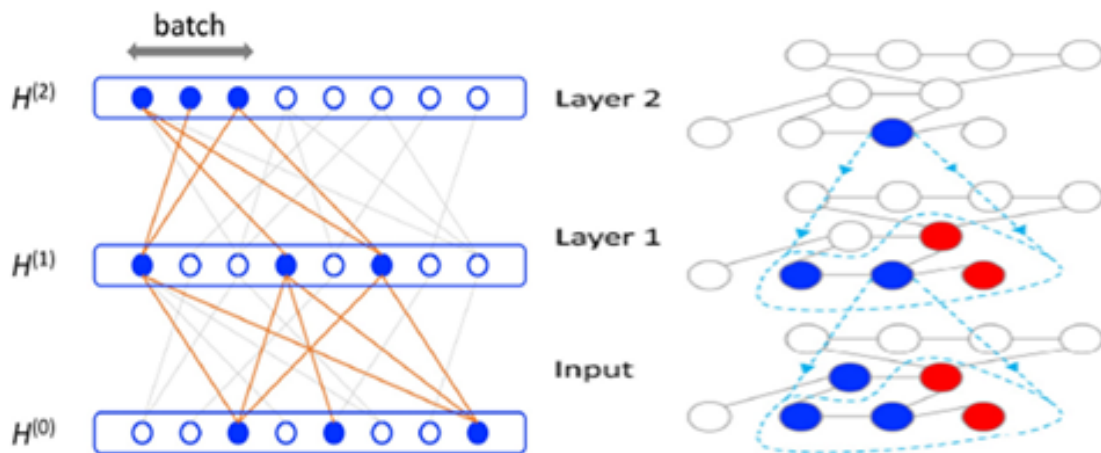
Many researches have observed that the most suitable depth for the existing GCNs is often very limited. This problem is potentially due to the practical difficulties involved in training deep GCNs or the over-smoothing problem, i.e., all nodes in deeper layers have the same representation. To remedy this problem, residual connections similar to ResNet can be added to GCNs. For example, Kipf and Welling added residual connections. [25]

III.4.3.3 Edge Features

The aforementioned GCNs mostly focus on utilizing node features and graph structures. but there is another important source of information: the edge features.

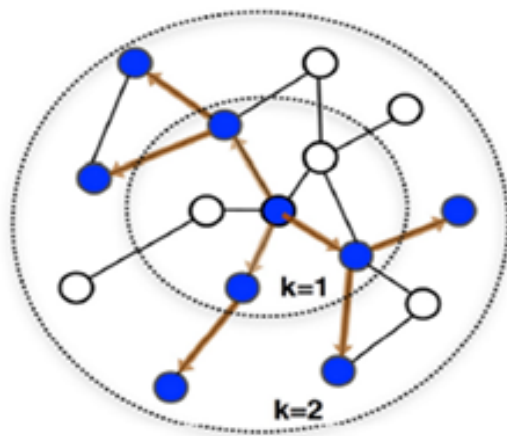
III.4.3.4 Sampling Methods

One critical bottleneck when training GCNs for large-scale graphs is efficiency. many GCNs follow a neighborhood aggregation scheme. However, because many real graphs follow a power-law distribution, the number of neighbors can expand extremely quickly. To deal with this problem, two types of sampling methods have been proposed: neighborhood samplings and layer-wise samplings, as illustrated in Figure (III.2).[25]

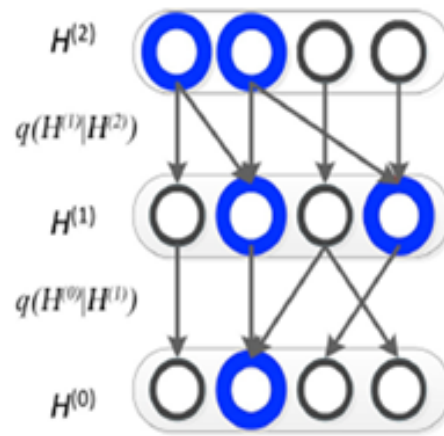


(A) the node sampling method in graphSAGE

(B) the node sampling method in stochastic GCN



(C) the node sampling method in fastGCN



(D) the node sampling method in adapt

Figure III.2: Different node sampling methods, in which the blue nodes indicate samples from one batch and the arrows indicate the sampling directions. The red nodes in (B) represent historical samples. [25]

III.4.3.5 Inductive Setting

Another important aspect of GCNs is that whether they can be applied to an inductive setting, i.e., training on a set of nodes or graphs and testing on another unseen set of nodes or graphs, the existing inductive GCNs is suitable only for graphs with explicit features. [25]

III.5 GRAPH AUTOENCODERS

The autoencoder (AE) and its variations have been widely applied in unsupervised learning tasks and are suitable for learning node representations for graphs. The implicit assumption is that graphs have an inherent, potentially nonlinear low-rank structure.

III.5.1 Autoencoders

The use of AEs for graphs originated from sparse autoencoder (SAE). The basic idea is that, by regarding the adjacency matrix or its variations as the raw features of nodes, AEs can be leveraged as a dimensionality reduction technique to learn lowdimensional node representations.

<i>Method</i>	<i>Type</i>	<i>Objective</i>	<i>T.C.</i>	<i>Node Features</i>	<i>Other Characteristics</i>
SAE	AE	L2-Reconstruction	$O(M)$	No	-
SDNE	AE	L2-Reconstruction + Laplacian Eigenmaps	$O(M)$	No	-
DNGR	AE	L2-Reconstruction	$O(N^2)$	No	-
GC-MC	AE	L2-Reconstruction	$O(M)$	Yes	GCN Encoder
DRNE	AE	Recursive Reconstruction	$O(N_s)$	No	LSTM Encoder
G2G	AE	KL + Ranking	$O(M)$	Yes	Nodes as distributions
VGAE	VAE	Pairwise Probability of Reconstruction	$O(N^2)$	Yes	GCN Encoder
DVNE	VAE	Wasserstein + Ranking	$O(M)$	No	Nodes as distributions
ARGA/ARVGA	AE/VAE	L2-Reconstruction + GAN	$O(N^2)$	Yes	GCN Encoder
NetRA	AE	Recursive Reconstruction + Laplacian Eigenmaps + GAN	$O(M)$	No	LSTM Encoder

Table III.5: A Comparison among Different Graph Autoencoders (GAEs). T.C. = Time Complexity. [25]

III.5.2 Variational Autoencoders

Different from the aforementioned autoencoders, variational autoencoders (VAEs) are another type of deep learning method that combines dimensionality reduction with generative models. Its potential benefits include tolerating noise and learning smooth representations. VAEs were first introduced to graph data in VGAE.[25]

III.5.3 Improvements and Discussions

Several improvements have also been proposed for GAEs.

III.5.3.1 Adversarial Training

An adversarial training scheme was incorporated into GAEs as an additional regularization term in ARGA. Specifically, the encoder of GAEs was used as the generator while the discriminator aimed to distinguish whether a latent representation came from the generator or from a prior distribution. In this way, the autoencoder was forced to match the prior distribution as a regularization.

III.5.3.2 Inductive Learning

Similar to GCNs, GAEs can be applied to the inductive learning setting. This can be achieved by using a GCN as the encoder, or by directly learning a mapping function. Because the edge

information is utilized only when learning the parameters, the model can also be applied to nodes unseen. These works also show that although GCNs and GAEs are based on different architectures, it is possible to use them jointly ” future direction”.

III.5.3.3 Similarity Measures

In GAEs, many similarity measures have been adopted, for example, L2-reconstruction loss, Laplacian eigenmaps, and the ranking loss for graph AEs, and KL divergence and Wasserstein distance for graph VAEs. Although these similarity measures are based on different motivations, how to choose an appropriate similarity measure for a given task and model architecture remains unstudied. [25]

III.6 GRAPH REINFORCEMENT LEARNING

One aspect of deep learning not yet discussed is reinforcement learning (RL), which has been shown to be effective in AI tasks such as playing games. RL is known to be good at learning from feedbacks, especially when dealing with non-differentiable objectives and constraints. In this section, we review Graph RL methods. Their main characteristics are summarized in Table (III.6).

<i>Method</i>	<i>Task</i>	<i>Actions</i>	<i>Rewards</i>	<i>Time Complexity</i>
GCPN	Graph generation	Link prediction	GAN + domain knowledge	$O(MN)$
MolGAN	Graph generation	Generate the whole graph	GAN + domain knowledge	$O(N^2)$
GTPN	Chemical reaction prediction	Predict node pairs and new bonds	Prediction results	$O(N^2)$
GAM	Graph classification	Predict graph labels and select the next node	Classification results	$O(d_{avg} sT)$
DeepPath	Knowledge graph reasoning	Predict the next node of the reasoning path	Reasoning results + diversity	$O(d_{avg} sT + s^2 T)$
MINERVA	Knowledge graph reasoning	Predict the next node of the reasoning path	Reasoning results	$O(d_{avg} sT)$

Table III.6: The Main Characteristics of Graph Reinforcement Learning. [25]

III.7 GRAPH ADVERSARIAL METHODS

Adversarial methods such as GANs and adversarial attacks have drawn increasing attention in the machine learning community in recent years. The main characteristics of graph adversarial methods are summarized in Table (III.7).

<i>Category</i>	<i>Method</i>	<i>Adversarial Methods</i>	<i>Time Complexity</i>	<i>Node Features</i>
Adversarial Training	ARGA/ARVGA	Regularization for GAEs	$O(N^2)$	Yes
	NetRA	Regularization for GAEs	$O(M)$	No
	GCPN	Rewards for Graph RL	$O(MN)$	Yes
	MolGAN	Rewards for Graph RL	$O(N^2)$	Yes
	GraphGAN	Generate negative samples (node pairs)	$O(MN)$	No
	ANE	Regularization for network embedding	$O(N)$	No
	GraphSGAN	Enhancing semi-supervised learning on graphs	$O(N^2)$	Yes
Adversarial Attack	NetGAN	Generate graphs via random walks	$O(M)$	No
	Nettack	Targeted attacks of graph structures and node attributes	$O(Nd_0^2)$	Yes
	Dai et al.	Targeted attacks of graph structures	$O(M)$	No
	Zugner and Gunnemann	Non-targeted attacks of graph structures	$O(N^2)$	Yes

Table III.7: The Main Characteristics of Graph Adversarial Methods. [25]

III.7.1 Adversarial Training

The basic idea behind a GAN is to build two linked models: a discriminator and a generator. The goal of the generator is to “fool” the discriminator by generating fake data, while the discriminator aims to distinguish whether a sample comes from real data or is generated by the generator. Subsequently, both models benefit from each other by joint training using a minimax game. Adversarial training has been shown to be effective in generative models and enhancing the generalization ability of discriminative models.[25]

III.7.2 Adversarial Attacks

Adversarial attacks are another class of adversarial methods intended to deliberately “fool” the targeted methods by adding small perturbations to data. Studying adversarial attacks can deepen our understanding of the existing models and inspire more robust architectures.

III.8 DISCUSSIONS AND CONCLUSION

Thus far, we have reviewed the different graph-based deep learning architectures as well as their similarities and differences. Next, we briefly discuss their applications, implementations, and future directions before summarizing this chapter.

III.8.1 Applications

In addition to standard graph inference tasks such as node or graph classification, graph-based deep learning methods have also been applied to a wide range of disciplines, including modeling social influence, recommendation, chemistry and biology, physics, disease and drug prediction gene expression, natural language processing (NLP), computer vision, traffic forecasting, program induction, solving graph-based NP problems, and multi-agent AI systems.[25]

III.8.2 Implementations

Recently, several open libraries have been made available for developing deep learning models on graphs. These libraries are listed in Table (III.8). We also collected a list of source code (mostly from their original authors) for the studies discussed in this chapter. These open implementations make it easy to learn, compare, and improve different methods. Some implementations also address the problem of distributed computing, which we do not discuss in this chapter. [25]

<i>Name</i>	<i>URL</i>	<i>Language/Framework</i>	<i>Key Characteristics</i>
PyTorch Geometric	https://github.com/rusty1s/pytorch_geometric	PyTorch	Improved efficiency, unified operations, comprehensive existing methods
Deep Graph Library	https://github.com/dmlc/dgl	PyTorch	Improved efficiency, unified operations, scalability
AliGraph	https://github.com/alibaba/aligraph	Unknown	Distributed environment, scalability, in-house algorithms
Euler	https://github.com/alibaba/euler	C++/TensorFlow	Distributed environment, scalability

Table III.8: Libraries of Deep Learning on Graphs. [25]

III.8.3 Future Directions

There are several ongoing or future research directions worth noting as well:

- New models for unstudied graph structures.
- Compositionality of existing models.
- Dynamics graphs.
- Interpretability and robustness.

III.9 Conclusion

The above information shows that deep learning on graphs is a promising and fast-developing research field that both offers exciting opportunities and presents many challenges. Studying deep learning on graphs constitutes a critical building block in modeling relational data, and it is an important step towards a future with better machine learning and artificial intelligence techniques. [25]

Chapter IV

Implementation and Evaluation

IV.1 INTRODUCTION

Given the ubiquity of network structures in real-world data, graph generative models have been studied extensively as a means of simulating graphs with different properties. modern approaches to graph generation based on deep learning, including graph convolutional networks [Kipf and Welling, 2016], Li et al., LGCN [2017], are flexible enough to learn multiple different properties of an input graph simultaneously. The graphs generated by these architectures may be used for downstream learning tasks such as data augmentation, recommendation, and link prediction.

In this chapter we made a special view on a graph convolutional networks approaches and we tried to implement three methods codes and compare their result , we tell about the tools that we need to execute the codes.

IV.2 Implementation framework

IV.2.1 Python

IV.2.1.1 What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.[26]

IV.2.1.2 It is used for

- Web development (server-side).
- Software development.
- Mathematics.
- System scripting.

IV.2.1.3 What can Python do

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.[26]

IV.2.1.4 Why Python ?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.[26]

IV.2.1.5 Good to know

- The most recent major version of Python is Python 3, which we shall be using in this work. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

IV.2.1.6 Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.[26]

IV.2.2 DATASETS

IV.2.2.1 CoRA Dataset

The experimental results have been carried out on a subset of the original CORA Research Paper Classification Dataset by Andrew McCallum of University of Massachusetts Amherst. The CORA dataset is composed by a set of entities and their relations to allow experimenting with machine learning approaches which can cope with relations. Entities are authors and scientific papers. CoRA assigns to each paper a set of categories (multi-label classification task), selected from a taxonomy of classes. The goal of our experiments is to predict the categories assigned to each paper[27].

IV.2.2.2 citeseer dataset

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

CiteSeer for Entity Resolution The CiteSeer dataset contains 1504 machine learning documents with 2892 author references to 165 author entities. For this dataset, the only attribute information available is author name. The full last name is always given, and in some cases the author's full first name and middle name are given and other times only the initials are given.[28]

IV.2.3 GOOGLE COLAB

IV.2.3.1 Colab definition

Colab is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user. The term is generally used to describe data centers available to many users over the Internet often have functions distributed

over multiple locations from central servers. If the connection to the user is relatively close, it may be designated an edge server.

Colab relies on sharing of resources to achieve coherence and economies of scale.

Advocates of **Colab** note that **Colab** function allows companies to avoid or minimize up-front IT infrastructure costs. Proponents also claim that **Colab** allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and that it enables IT teams to more rapidly adjust resources to meet fluctuating and unpredictable demand, providing the burst computing capability: high computing power at certain periods of peak demand[29].

IV.2.3.2 How does colab work?

Rather than owning their own computing infrastructure or data centers, companies can rent access to anything from applications to storage from a cloud service provider.

One benefit of using google **Colab** services is that firms can avoid the upfront cost and complexity of owning and maintaining their own IT infrastructure, and instead simply pay for what they use, when they use it.

In turn, providers of **Colab** services can benefit from significant economies of scale by delivering the same services to a wide range of customers.

IV.2.3.3 What colab services are available?

Google Colab services cover a vast range of options now, from the basics of storage, networking, and processing power through to natural language processing and artificial intelligence as well as standard office applications. Pretty much any service that doesn't require you to be physically close to the computer hardware that you are using can now be delivered via the cloud [29].

IV.3 IMPLEMENTATION

IV.3.1 Semi-Supervised Classification with GCNs "Li et al" method

In this part, we demystify the GCN model for semisupervised learning. In particular, we show that the graph convolution of the GCN model is simply a special form of Laplacian smoothing, which mixes the features of a vertex and its nearby neighbors. The smoothing operation makes the features of vertices in the same cluster similar, thus greatly easing the classification task, which is the key reason why GCNs work so well. However, it also brings potential concerns of over-smoothing. If a GCN is deep with many convolutional layers, the output features may be oversmoothed and vertices from different clusters may become indistinguishable. The mixing happens quickly on small datasets with only a few convolutional layers. Also, adding more layers to a GCN will make it much more difficult to train. However, a shallow GCN model such as the two-layer GCN used in (Kipf and Welling 2017)[30] has its own limits. Besides that, it requires many additional labels for validation, it also suffers from the localized nature of the convolutional filter. When only few labels are given, a shallow GCN cannot effectively propagate the labels to the entire data graph. the performance of GCNs drops quickly as the training size shrinks, even for the one with 500 additional labels for validation. To overcome the limits and realize the full potentials of the GCN model, we propose a co-training approach and a self-training approach to train GCNs. By co-training a GCN with a random walk model, the latter could complement the former in exploring global graph topology. By self-training a GCN,

we can exploit its feature extraction capability to overcome its localized nature. Combining both the co-training and self-training approaches can substantially improve the GCN model for semi-supervised learning with very few labels, and exempt it from requiring additional labeled data for validation. our method outperforms GCNs by a large margin.

IV.3.1.1 dataset used

<i>Dataset</i>	<i>Nodes</i>	<i>Edges</i>	<i>Classes</i>	<i>Features</i>
CiteSeer	3327	4732	6	3703
Cora	2708	5429	7	1433

Table IV.1: Dataset statistics. [32]

IV.3.1.2 Results Analysis

The classification results are summarized in this Tables, where the highest accuracy in each column is highlighted in bold and the top 3 are underlined. Our methods are displayed at the bottom half of each table. We can see that the performance of Co-Training is closely related to the performance of LP. If the data has strong manifold structure, such as PubMed, Co-Training performs the best. In contrast, Self-Training is the worst on PubMed, as it does not utilize the graph structure. But Self-Training does well on CiteSeer where Co-Training is overall the worst. Intersection performs better when the training size is relatively large, because it filters out many labels. Union performs best in many cases since it adds more diverse labels to the training set [32].

Cora						
<i>Label Rate</i>	0.5%	1%	2%	3%	4%	5%
<i>LP</i>	<u>56.4</u>	62.3	65.4	67.5	69.0	70.2
<i>Cheby</i>	38.0	52.0	62.4	70.8	74.1	77.6
<i>GCN-V</i>	42.6	56.9	67.8	74.9	77.6	79.3
<i>GCN+V</i>	50.9	62.3	72.2	76.5	78.4	79.7
<i>Co-training</i>	<u>56.6</u>	<u>66.4</u>	<u>73.5</u>	75.9	78.9	<u>80.8</u>
<i>Self-training</i>	53.7	<u>66.1</u>	<u>73.8</u>	<u>77.2</u>	<u>79.4</u>	80.0
<i>Union</i>	<u>58.5</u>	<u>69.9</u>	<u>75.9</u>	<u>78.5</u>	<u>80.4</u>	<u>81.7</u>
<i>Intersection</i>	49.7	65.0	72.9	<u>77.1</u>	<u>79.4</u>	<u>80.2</u>

Table IV.2: Classification Accuracy On Cora. [32]

<i>Label Rate</i>	CiteSeer					
	0.5%	1%	2%	3%	4%	5%
<i>LP</i>	34.8	40.2	43.6	45.3	46.4	47.3
<i>Cheby</i>	31.7	42.8	59.9	66.2	68.3	69.3
<i>GCN-V</i>	33.4	46.5	62.6	66.9	68.4	69.5
<i>GCN+V</i>	43.6	55.3	64.9	67.5	68.7	69.6
<i>Co-training</i>	<u>47.3</u>	<u>55.7</u>	<u>62.1</u>	<u>62.5</u>	<u>64.5</u>	<u>65.5</u>
<i>Self-training</i>	43.3	<u>58.1</u>	<u>68.2</u>	<u>69.8</u>	<u>70.4</u>	<u>71.0</u>
<i>Union</i>	46.3	<u>59.1</u>	<u>66.7</u>	<u>66.7</u>	<u>67.6</u>	<u>68.2</u>
<i>Intersection</i>	<u>42.9</u>	<u>59.1</u>	<u>68.6</u>	<u>70.1</u>	<u>70.8</u>	<u>71.2</u>

Table IV.3: Classification Accuracy on CiteSeer. [25]

IV.3.1.3 Comparison with other methods

We compare the GCNs methods with other state-of-the-art methods in Table IV.4 . The experimental setup is the same except that for every dataset, we sample 20 labels for each class, which corresponds to the total labeling rate of 3.6% on CiteSeer, 5.1% on Cora. The results of other baselines are copied from (Kipf and Welling 2017)[30] . Our methods perform similarly as GCNs and outperform other baselines significantly. Although we did not directly compare with other baselines, we can see from Table IV.2, IV.3 that our methods with much fewer labels already outperform many baselines. For example, our method Union on Cora Table (IV.3) with 2% labeling rate (54 labels) beats all other baselines with 140 labels Table (IV.4).[32]

Method	<i>CiteSeer</i>	<i>Cora</i>
<i>ManiReg</i>	60.1	59.5
<i>SemiEmb</i>	59.6	59.0
<i>LP</i>	45.3	68.0
<i>DeepWalk</i>	43.2	67.2
<i>ICA</i>	<u>69.1</u>	75.1
<i>Planetoid</i>	64.7	75.7
GCN-V	68.1	80.0
GCN+V	<u>68.9</u>	<u>80.3</u>
Co-training	64.0	79.6
Self-training	67.8	<u>80.2</u>
Union	65.7	<u>80.5</u>
Intersection	<u>69.9</u>	79.8

Table IV.4: Accuracy under 20 Labels per Class. [32]

IV.3.2 Graph convolutional network: kiph and willing method

We present a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs. We motivate the choice of our convolutional architecture via a localized first-order approximation of spectral graph convolutions. Our model scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes. In a number of experiments on citation networks and on a knowledge graph dataset we demonstrate that our approach outperforms related methods by a significant margin.[30]

IV.3.2.1 SEMI-SUPERVISED NODE CLASSIFICATION

we can relax certain assumptions typically made in graph-based semi-supervised learning by conditioning our model $f(X;A)$ both on the data X and on the adjacency matrix A of the underlying graph structure. We expect this setting to be especially powerful in scenarios where the adjacency matrix contains information not present in the data X , such as citation links between documents in a citation network or relations in a knowledge graph. The overall model, a multi-layer GCN for semi-supervised learning

IV.3.2.2 GRAPH-BASED SEMI-SUPERVISED LEARNING

A large number of approaches for semi-supervised learning using graph representations have been proposed in recent years, most of which fall into two broad categories: methods that use some form of explicit graph Laplacian regularization and graph embedding-based approaches. Prominent examples for graph Laplacian regularization include label propagation, manifold regularization and deep semi-supervised embedding.

Recently, attention has shifted to models that learn graph embeddings with methods inspired by the skip-gram model. DeepWalk learns embeddings via the prediction of the local neighborhood of nodes, sampled from random walks on the graph. LINE and node2vec extend DeepWalk with more sophisticated random walk or breadth-first search schemes. For all these methods, however, a multistep pipeline including random walk generation and semi-supervised training is required where each step has to be optimized separately. Planetoid alleviates this by injecting label information in the process of learning embeddings.

IV.3.2.3 EXPERIMENTS

We test our model in a number of experiments: semi-supervised document classification in citation networks, semi-supervised entity classification in a bipartite graph extracted from a knowledge graph, an evaluation of various graph propagation models and a run-time analysis on random graphs.

IV.3.2.4 DATASETS

We closely follow the experimental setup in Yang et al. (2016). Dataset statistics are summarized in Table IV.5. In the citation network datasets—Citeseer, Cora and Pubmed (Sen et al., 2008)—nodes are documents and edges are citation links. Label rate denotes the number of labeled nodes that are used for training divided by the total number of nodes in each dataset. NELL (Carlson et al., 2010; Yang et al., 2016) is a bipartite graph dataset extracted from a knowledge graph with 55,864 relation nodes and 9,891 entity nodes.

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Table IV.5: Dataset statistics, as reported in Yang et al. (2016). [32]

IV.3.2.5 RESULTS

SEMI-SUPERVISED NODE CLASSIFICATION: Results are summarized in Table IV.6. Reported numbers denote classification accuracy in percent. For ICA, we report the mean accuracy of 100 runs with random node orderings. Results for all other baseline methods are taken from the Planetoid paper (Yang et al., 2016). Planetoid* denotes the best model for the respective dataset out of the variants presented in their paper.

IV.3.2.6 Results Analysis

This approach introduced a novel approach for semi-supervised classification on graph-structured data. Our GCN model uses an efficient layer-wise propagation rule that is based on a first-order approximation of spectral convolutions on graphs. Experiments on a number of network

Method	Citeseer	Cora	Pubmed	NELL
ManiReg	60.1	59.5	70.7	21.8
SemiEmb	59.6	59.0	71.1	26.7
LP	45.3	68.0	63.0	26.5
DeepWalk	43.2	67.2	65.3	58.1
ICA	69.1	75.1	73.9	23.1
Planetoid*	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	70.3 (7s)	81.5 (4s)	79.0 (38s)	66.0 (48s)
GCN (rand. splits)	67.9 \pm 0.5	80.1 \pm 0.5	78.9 \pm 0.7	58.4 \pm 1.7

Table IV.6: Summary of results in terms of classification accuracy (in percent).

datasets suggest that the proposed GCN model is capable of encoding both graph structure and node features in a way useful for semi-supervised classification. In this setting, our model outperforms several recently proposed methods by a significant margin, while being computationally efficient.

IV.3.3 graph convolutional network : " LGCN method "

IV.3.3.1 introduction

Deep learning methods are becoming increasingly powerful in solving various challenging artificial intelligence tasks. Among these deep learning methods, convolutional neural networks (CNNs) have demonstrated promising performance in many image-related applications, such as image classification, semantic segmentation, and object detection. A variety of CNN models have been proposed to continuously set the performance records. In addition to images, CNNs have also been successfully applied to natural language processing tasks such as neural machine translation. One common characteristic behind these tasks is that the data can be represented by grid-like structures. This enables the use of convolutional operations in the form of the same local filters scanning every position on the input. Unlike traditional hand-crafted filters, the local filters used in convolutional layers are trainable. The networks can automatically decide what kind of features to extract by learning the weights in these trainable filters, thereby avoiding hand-crafted feature extraction[31].

IV.3.3.2 METHODS

In this section, we introduce the learnable graph convolutional layer (LGCL) and the sub-graph training strategy on generic graph data. Based on these developments, we propose the large-scale learnable graph convolutional networks (LGCNs).

IV.3.3.3 EXPERIMENTAL STUDIES

In this section, we evaluate this proposed large-scale learnable graph convolutional networks (LGCNs) on node classification tasks under both transductive and inductive learning settings. In addition to comparisons with prior state-of-the-art models, some performance studies are performed to investigate how to choose hyperparameters. Experiments are also conducted to analyze the training strategy based on the proposed sub-graph selection algorithm. Experimental results show that LGCNs yield improved performance, and the sub-graph training is much more efficient than whole-graph training. this code is publicly available.

Dataset	#Nodes	#Features	#Classes	#Training Nodes	#Validation Nodes	#Test Nodes	Degree	Setting
Cora	2708	1433	7	140	500	1000	4	Transductive
Citeseer	3327	3703	6	120	500	1000	5	Transductive
Pubmed	19717	500	3	60	500	1000	6	Transductive
PPI	56944	50	121	44906 (20 graphs)	6514 (2 graphs)	5524 (2 graphs)	31	Inductive

Table IV.7: Summary of datasets used in our experiments [30, 31]. The Cora, Citeseer, and Pubmed datasets are used for transductive learning experiments, while the PPI dataset is for inductive learning experiments. The degree attribute listed is the average node degree of each dataset, which helps the selection of the hyper-parameter k in LGCLs. [31]

Experimental Setup

We describe the experimental setup under both transductive and inductive learning settings. **Transduction Learning.** In transductive learning tasks, we employ the proposed LGCN models

Since transductive learning datasets employ high-dimensional bag-of-words representations as feature vectors of nodes, the inputs go through a graph embedding layer to reduce the dimension. Here, we use a GCN layer as the graph embedding layer. The dimension of the embedding output is 32. Then we apply LGCLs, each of which uses $k = 8$ and produces 8-component feature vectors. For the Cora, Citeseer, and Pubmed, we stack 2, 1, and 1 LGCLs, respectively. We use concatenation in skip connections. Finally, a fully-connected layer is used as a classifier to make predictions. Before the fully-connected layer, we perform a simple sum to aggregate feature vectors of adjacent nodes. Dropout is applied on both input feature vectors and adjacency matrices in each layer with rates of 0.16 and 0.999, respectively. All LGCN models in transductive learning tasks use the sub-graph training strategy. The sub-graph size is set to 2,000.

Inductive Learning. For inductive learning, the same LGCN model is used except for some hyper-parameters. For the graph embedding layer, the dimension of output feature vectors is 128. We stack two LGCLs with $k = 64$. We also employ the subgraph training strategy, with sub-graph initial node size equal to 500 and 200. Dropout with a rate of 0.9 is applied in each layer. For both transductive and inductive learning LGCN models, the following configurations are shared. For all layers, only the identity activation function is used, which means no nonlinearity is involved in the networks. In order to avoid over-fitting, the L2 regularization with $\lambda = 0.0005$ is applied. For training, the Adam optimizer with a learning rate of 0.1 is used. Weights in LGCNs are initialized by the Glorot initialization. We employ the early stopping strategy based on the validation accuracy and train 1,000 epochs at most.

IV.3.3.4 Results Analysis

The experimental results are summarized in Tables IV.8 and IV.9 for transductive and learning settings, respectively.

Transduction Learning For transductive learning experiments, we report node classification accuracies. Table IV.8 provides the comparisons with other graph models. According to the results, our LGCN models achieve better performance over the current state-of-the-art GCNs by a margin of 1.8%, 2.7%, and 0.6% on the Cora, Citeseer, and Pubmed datasets,

		Cora	Citeseer	Pubmed
GCN	# Nodes	2708	3327	19717
	Accuracy	81.5%	70.3%	79.0%
	Time	7s	4s	38s
$LGCN_{whole}$	# Nodes	2708	3327	19717
	Accuracy	$83.8 \pm 0.5\%$	$73.0 \pm 0.6\%$	$79.5 \pm 0.2\%$
	Time	58s	30s	1080s
$LGCN_{sub}$	# Nodes	644	442	354
	Accuracy	$83.3 \pm 0.5\%$	$73.0 \pm 0.6\%$	$79.5 \pm 0.2\%$
	Time	14s	3.6s	2.6s

Table IV.8: Results of transductive learning experiments for comparing the sub-graph training and whole-graph training strategies on the Cora, Citeseer, and Pubmed datasets. For comparison, we conduct experiments on LGCNs that employ the same whole-graph training strategy as GCNs, denoted as $LGCN_{whole}$. [31]

respectively.

Inductive Learning For inductive learning experiments, we report micro-averaged F1 scores. From table IV.9, we can observe that our LGCN model outperforms GraphSAGE-LSTM by a margin of 16%. Without observing the structure of test graphs in training, the LGCN model still achieves good generalization. The results above show that the proposed LGCN models on generic graphs consistently yield new state-of-the-art performance in node classification tasks on different datasets. These results demonstrate the effectiveness of applying regular convolutional operations on transformed graph data. In addition, the proposed transformation approach through the k-largest node selection is shown to be effective.

IV.3.3.5 abstract

In this work, we tested the learnable graph convolutional layer(LGCL), which transforms generic graphs to data of grid-like structures and enables the use of regular convolutional operations. The transformation is conducted through a novel k-largest node selection process, which uses the ranking between node feature values. Based on our LGCL, we build deeper networks, known as learnable graph convolutional networks (LGCNs), for node classification tasks on graphs. Experimental results show that the proposed LGCN models yield consistently better performance than prior methods under both transductive and inductive learning settings. The LGCN models achieve new state-of-the-art results on four different datasets, demonstrating the effectiveness of LGCLs.

Models	PPI
GraphSAGE-GCN [9]	0.500
GraphSAGE-mean [9]	0.598
GraphSAGE-pool [9]	0.600
GraphSAGE-LSTM [9]	0.612
LGCNsub(Ours)	0.772 ± 0.002

Table IV.9: Results of transductive learning experiments for comparing the LGCNsub and GCN layers on the Cora, Citeseer, and Pubmed datasets. Using the network architecture of LGCNsub , we replace LGCLs by GCN layers, resulting in the LGCNsub -GCN model. [31]

Models	Cora	Citeseer	Pubmed
$LGCN_{sub}$ -GCN	$82.2 \pm 0.5\%$	$71.1 \pm 0.5\%$	$79.0 \pm 0.2\%$
$LGCN_{sub}$ (Ours)	$83.3 \pm 0.5\%$	$73.0 \pm 0.6\%$	$79.5 \pm 0.2\%$

Table IV.10: Results of transductive learning experiments for comparing the $LGCN_{sub}$ and GCN layers on the Cora, Citeseer, and Pubmed datasets. Using the network architecture of $LGCN_{sub}$, we replace LGCLs by GCN layers, resulting in the $LGCN_{sub}$ -GCN model. [31]

IV.4 CONCLUSION

Thus far, we have reviewed the different approaches to deep learning structures ” graph convolutional network ” along with their similarities and differences after applying three of them to the same dataset. We investigated the results and showed them. With all this, work is still under-way in this area to reach more efficient deep learning methods, by using the compare between the result we can know all the deffrences between and can find the most accuracy methode in every single dataset that can give the most result is effecient.

General Conclusion

Graphs are common data structures used to represent / model real-world systems and the data associated with it. Graph Mining is one of the arms of Data mining in which voluminous complex data are represented in the form of graphs and mining is done to infer knowledge from them. This work was focused in the particular problem of frequent subgraph mining.

Frequent sub graph mining or for short (FSM) is extensively used for graph classification, building indices and graph clustering purposes, the purpose of an FSM algorithm is to discover subgraphs that appears frequently in a set of graphs or a single large graph.

The deep learning is a big topic that can deal correctly and smart with graph " frequent subgraph mining " that have a lot of categories and methods, in our work we covered the convolutional neural network and a three methods from the most efficient, then we compare our result between it each other, finally we can say that This field is subject to development, and its methods can be applied in all areas of life to improve results and make them more accurate.

References

- [1] Aridhi S. *Distributed Frequent Subgraph Mining in the Cloud*. [Engineering Doctoral School of Clermont Ferrand]; 2013.
- [2] Müller DE. *Lesematerial Big Data Analytics*. openHPI [Internet]. Available from: <https://open.hpi.de/courses/bigdata2017/items/4T09c3oDe1AhP9tecEUDp5> cited 2018 May 25.
- [3] Washio T, Motoda H. *State of the art of graph-based data mining*. ACM SIGKDD Explor Newsl [Internet]. [2003;5(1):59]. Available from: <http://portal.acm.org/citation.cfm?doid=959242.959249> July 2003.
- [4] Davide Mottin, Konstantina Lazaridou *Graph Mining: Introduction*. [GRAPH MINING WS 2016]. Available from: https://hpi.de/fileadmin/user_upload/fachgebiete/mueller/courses/graphmining/GraphMining-01-Introduction.pdf ,University of Potsdam.
- [5] *Graph Types and Applications*. [Internet]. Available from: <https://www.geeksforgeeks.org/graph-types-and-applications/> Last Updated: 16-11-2018.
- [6] Chuntao Jiang FC and MZ. *A Survey of Frequent Subgraph Mining Algorithms*. *Knowl Eng Rev*. 2004;00(January):1–24.
- [7] Gutman I, Polansky OE. *Mathematical concepts in organic chemistry*. Springer Science & Business Media; 2012. 28 p.
- [8] Aridhi S. *Distributed Frequent Subgraph Mining in the Cloud*. *Engineering Doctoral School of Clermont Ferrand*;2013.
- [9] B Rajinikanth. *Data Structures*. http://www.btechsmartclass.com/data_structures/tree-terminology.html (August 2020)
- [10] Ullmann JR. *An algorithm for subgraph isomorphism*. *J ACM*. 23(1):31–42. (1976)
- [11] Schmidt DC, Druffel LE. *A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices*. *J ACM*. 1976;23(3):433–45
- [12] McKay BD, others. *Practical graph isomorphism*. 1981;45–87.
- [13] Conte D, Foggia P, Sansone C, Vento M. *Thirty years of graph matching in pattern recognition*. *Int J pattern Recognit Artif Intell*. 2004;18(03):265–98.

- [14] Miyazaki T. *The complexity of McKays canonical labeling algorithm*. In: *Groups and Computation II*. 1997. p. 239–56
- [15] Cordella LP, Foggia P, Sansone C, Vento M. *Subgraph transformations for the inexact matching of attributed relational graphs*. In: *Graph based representations in pattern recognition*. Springer; 1998. p. 43–52.
- [16] Cordella L Pietro, Foggia P, Sansone C, Vento M. *An improved algorithm for matching large graphs*. In: *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*. 2001. p. 149–59.
- [17] Mrzic, Aida and Meysman, Pieter and Bittremieux, Wout and Moris, Pieter and Cule, Boris and Goethals, Bart and Laukens, Kris *Grasping frequent subgraph mining for bioinformatics applications*. (German) [*On the electrodynamics of moving bodies*]. Annalen der Physik, 2018
- [18] Krish. *graph-mining*.
https://www.slideshare.net/Krish_ver2/55-graph-mining
(May 7, 2015)
- [19] Darren Rolfe, Vince. *Graph Mining and Social Network Analysis Data Mining*. In: 11.06.14.
- [20] *Difference Between Fp growth and Apriori Algorithm*.
<http://www.lastnightstudy.com/Show?id=123/Difference-Between-Fp-growth-and-Apriori-Algorithm> (August, 2020)
- [21] Ramraj, T and Prabhakar, R. *Frequent subgraph mining algorithms-a survey*. 2015; 47:197–204.
- [22] Tutorials Point. *Artificial Intelligence*. Retrieved from
https://www.dcehvp.com/EContent/BCA/BCA-III/artificial_intelligence_tutorial.pdf
(2015).
- [23] Tutorials Point. (2015). *Machine Learning*. Retrieved from
https://www.tutorialspoint.com/machine_learning/machine_learning_tutorial.pdf
(2019)
- [24] STEFFORA MUTSCHLER *Difference Between Fp growth and Apriori Algorithm*.
<https://semiengineering.com/using-cnns-to-speed-up-systems/> . AUGUST 10TH, 2017.
- [25] Zhang, Ziwei and Cui, Peng and Zhu, Wenwu. *Deep learning on graphs: A survey*. 2020.
- [26] Python Tutorial. *Python Introduction*.
Retrieved from
https://www.w3schools.com/python/python_intro.asp
(2020).
- [27] Claudio Saccà. *Experiments on CoRA Dataset*.
Retrieved from
https://sites.google.com/site/semanticbasedregularization/home/software/experiments_on_cora Jun 23, 2012, 10:06 AM.

- [28] University of California. *Datasets*.
Retrieved from
<https://links.soe.ucsc.edu/data>
UC Santa Cruz, 1156 High Street, Santa Cruz, CA 95064(2020).
- [29] Africa's Premium Managed. *Cloud Computing*.
Retrieved from
<https://addicnet.com/cloud-computing-2/>
December 9th, 2019.
- [30] Kipf, Thomas N and Welling, Max. *Semi-supervised classification with graph convolutional networks*.
arXiv preprint arXiv:1609.02907,(2016).
- [31] Gao, Hongyang and Wang, Zhengyang and Ji, Shuiwang. *Large-scale learnable graph convolutional networks*.
(2018):1416–1424.
- [32] Li, Qimai and Han, Zhichao and Wu, Xiao-Ming. *Deeper insights into graph convolutional networks for semi-supervised learning*.
arXiv preprint arXiv:1801.07606,(2018).
- [33] Ahmed, Abdelkader Ralem and Boubekour, Abdellatif. *Submitted to the computer science departement faculty of science and TECHNOLOGY-KHEMIS MILIANA UNIVERSITY in partial fulfillment of the requirement for the degree of master in computer science*.
(2018).
- [34] Joget. *A Quick Introduction to Artificial Intelligence, Machine Learning, Deep Learning and TensorFlow*.
(March 1st 2019).

List of Abbreviations

AI	Artificial Intellegance
FSM	Frequent Subgraph Mining
KDD	Knowledge Discovery in Databases
BFS	Breadth First Search
DFS	Depth First Search
DCP	Downward Closure Property
ANN	Artificial Neural Network
HPC	High Performance Computing
IDE	Integrated Development Envirenement
GCN	Graph Convolutional Network
CNN	Convolutional Neural Network
GNN	Graph Convolutional Network
DP	Deep Learning
DRL	Deep Reinforcement Learning
KNN	K-Nearest Neighbour
CPU	Central Processing Unit
ML	Machine Learning
GGs-NNs	Gated Graph Sequence Neural Networks
DGNN	Dynamic Graph Neural Network
LSTM	Long Short Term Memory
MPNNs	Message Passing Neural Network
GAN	Graph Attention Network
AE	Auto Encoder
NLD	Natural Language Processing
LGCL	Learnable Graph Convolutional Layer

Appendices

In this part we want to show some of the steps of executing our code of implementation and the result of every single method.

STEP 1: take a copy of the codes from the github site.

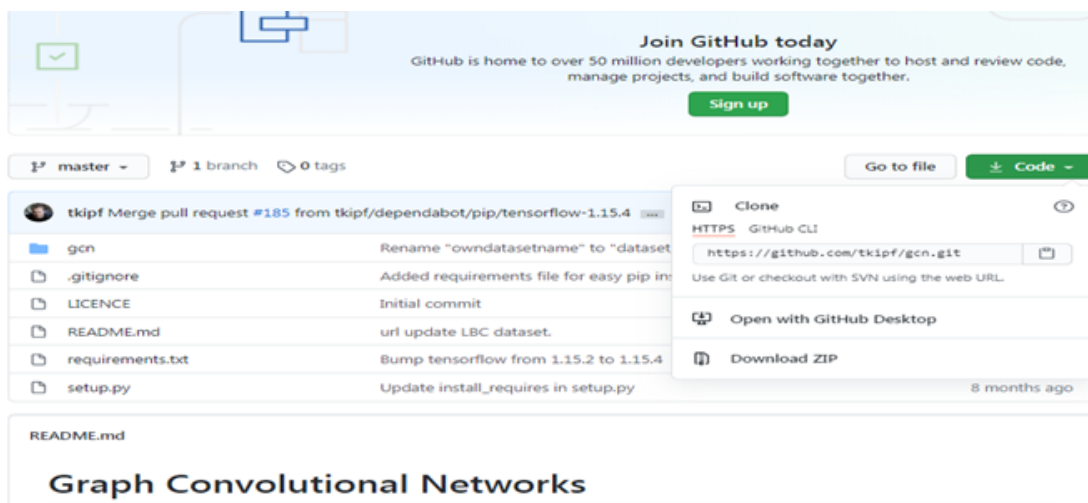


Figure .1: Take a copy code from github for kiph and welling method of GCNN.

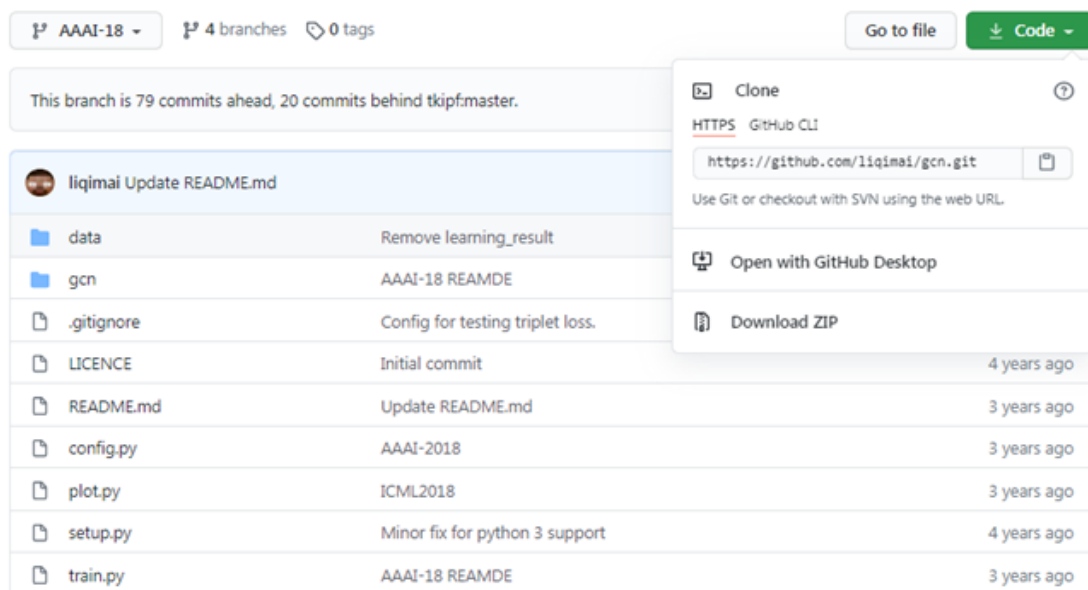


Figure .2: Take a copy code from github for li et al method of GCNN.

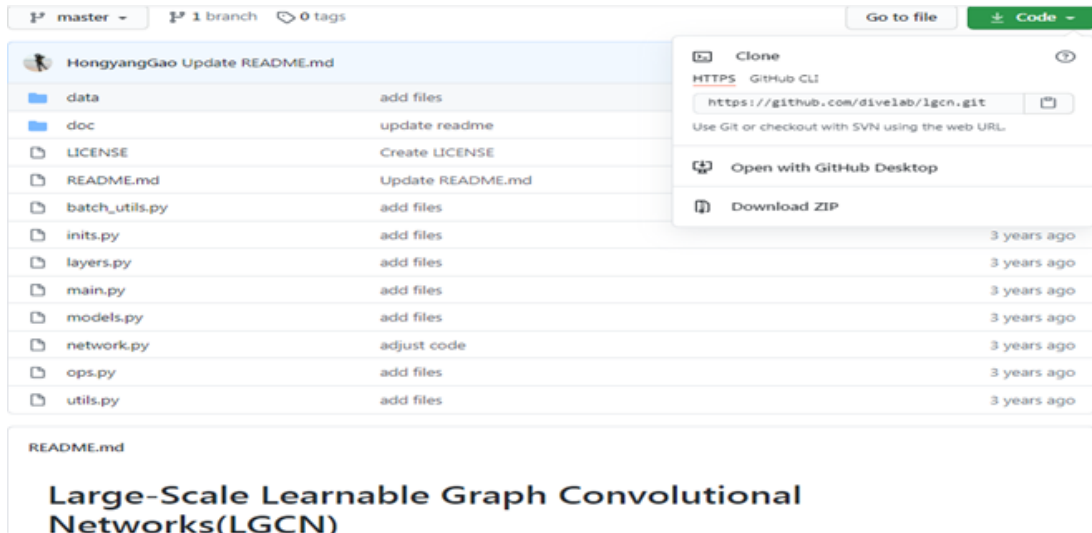


Figure .3: Take a copy code from github for LGCN method of GCNN.

STEP 2: Import the codes in the colab in a new notebook.

```
! git clone https://github.com/divelab/lgcn.git
```

```
Cloning into 'lgcn'...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 60 (delta 12), reused 13 (delta 4), pack-reused 28
Unpacking objects: 100% (60/60), done.
```

Figure .4: Import LGCN method code.

```
!git clone https://github.com/tkipf/gcn.git
```

```
Cloning into 'gcn'...
remote: Enumerating objects: 162, done.
remote: Total 162 (delta 0), reused 0 (delta 0), pack-reused 162
Receiving objects: 100% (162/162), 5.08 MiB | 25.15 MiB/s, done.
Resolving deltas: 100% (80/80), done.
```

Figure .5: Import Kipth and Welling method code .

```
!git clone https://github.com/liqimai/gcn.git

Cloning into 'gcn'...
remote: Enumerating objects: 515, done.
remote: Total 515 (delta 0), reused 0 (delta 0), pack-reused 515
Receiving objects: 100% (515/515), 51.81 MiB | 24.58 MiB/s, done.
Resolving deltas: 100% (339/339), done.
```

Figure .6: Import Li et Al method code.

STEP 3: follow the instruction in the GitHub site and read the informations , the find a solution of every single error for make a run.

```
!python main.py
step: 308 --- loss: 1.5580, train: 0.780, val: 0.778
step: 309 --- loss: 1.6142, train: 0.743, val: 0.780
step: 310 --- loss: 1.4451, train: 0.807, val: 0.776
step: 311 --- loss: 1.6420, train: 0.764, val: 0.780
step: 312 --- loss: 1.5855, train: 0.750, val: 0.774
step: 313 --- loss: 1.5955, train: 0.750, val: 0.768
step: 314 --- loss: 1.5861, train: 0.736, val: 0.754
step: 315 --- loss: 1.6210, train: 0.750, val: 0.712
step: 316 --- loss: 1.5637, train: 0.771, val: 0.756
step: 317 --- loss: 1.4931, train: 0.793, val: 0.758
step: 318 --- loss: 1.5377, train: 0.786, val: 0.770
step: 319 --- loss: 1.4715, train: 0.821, val: 0.776
step: 320 --- loss: 1.3833, train: 0.807, val: 0.780
step: 321 --- loss: 1.4682, train: 0.793, val: 0.764
step: 322 --- loss: 1.5309, train: 0.750, val: 0.762
step: 323 --- loss: 1.5960, train: 0.743, val: 0.784
step: 324 --- loss: 1.5539, train: 0.800, val: 0.776
step: 325 --- loss: 1.5528, train: 0.750, val: 0.770
step: 326 --- loss: 1.5369, train: 0.757, val: 0.758
step: 327 --- loss: 1.5209, train: 0.786, val: 0.772
step: 328 --- loss: 1.3930, train: 0.793, val: 0.772
step: 329 --- loss: 1.6286, train: 0.779, val: 0.746
```

Figure .7: LGCN run result.

```
!python train.py
Epoch: 0147 train_loss= 0.66998 train_acc= 0.96429 val_loss= 1.13289 val_acc= 0.78200 time= 0.00980
Epoch: 0148 train_loss= 0.67441 train_acc= 0.97143 val_loss= 1.13132 val_acc= 0.78200 time= 0.01028
Epoch: 0149 train_loss= 0.67382 train_acc= 0.95000 val_loss= 1.13003 val_acc= 0.78200 time= 0.00919
Epoch: 0150 train_loss= 0.64519 train_acc= 0.95714 val_loss= 1.12849 val_acc= 0.78200 time= 0.00955
Epoch: 0151 train_loss= 0.67857 train_acc= 0.96429 val_loss= 1.12742 val_acc= 0.78200 time= 0.00889
Epoch: 0152 train_loss= 0.70058 train_acc= 0.97143 val_loss= 1.12659 val_acc= 0.78200 time= 0.00955
Epoch: 0153 train_loss= 0.72587 train_acc= 0.94286 val_loss= 1.12525 val_acc= 0.78200 time= 0.00939
Epoch: 0154 train_loss= 0.66385 train_acc= 0.97857 val_loss= 1.12345 val_acc= 0.78200 time= 0.00963
Epoch: 0155 train_loss= 0.68267 train_acc= 0.96429 val_loss= 1.12172 val_acc= 0.78200 time= 0.00950
Epoch: 0156 train_loss= 0.68375 train_acc= 0.99286 val_loss= 1.12038 val_acc= 0.78200 time= 0.00933
Epoch: 0157 train_loss= 0.61288 train_acc= 0.97857 val_loss= 1.11847 val_acc= 0.78000 time= 0.00937
Epoch: 0158 train_loss= 0.64976 train_acc= 0.96429 val_loss= 1.11592 val_acc= 0.78000 time= 0.00935
Epoch: 0159 train_loss= 0.62890 train_acc= 0.95714 val_loss= 1.11343 val_acc= 0.78000 time= 0.00943
Epoch: 0160 train_loss= 0.68008 train_acc= 0.96429 val_loss= 1.11095 val_acc= 0.78000 time= 0.00923
Epoch: 0161 train_loss= 0.66698 train_acc= 0.97143 val_loss= 1.10845 val_acc= 0.78200 time= 0.00987
Epoch: 0162 train_loss= 0.66046 train_acc= 0.95714 val_loss= 1.10532 val_acc= 0.78200 time= 0.00899
Epoch: 0163 train_loss= 0.64797 train_acc= 0.96429 val_loss= 1.10240 val_acc= 0.77800 time= 0.00934
Epoch: 0164 train_loss= 0.65257 train_acc= 0.93571 val_loss= 1.10020 val_acc= 0.77800 time= 0.00926
Epoch: 0165 train_loss= 0.70419 train_acc= 0.94286 val_loss= 1.09817 val_acc= 0.77800 time= 0.00970
Epoch: 0166 train_loss= 0.69182 train_acc= 0.97143 val_loss= 1.09627 val_acc= 0.77800 time= 0.01172
Epoch: 0167 train_loss= 0.64690 train_acc= 0.97143 val_loss= 1.09485 val_acc= 0.77800 time= 0.00960
```

Figure .8: Kipth and welling run result .

```
!python train.py
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
* https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md
* https://github.com/tensorflow/addons
* https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.

Namespace(dataset=None, repeating=None, train_size=None, verbose=False)
{ 'default': { 'Model': 0,
              'Model19': 'union',
              'Model_to_add_label': {'Model': 0},
              'Model_to_predict': {'Model': 0},
              'alpha': 1e-06,
              'connection': 'cc',
              'conv': 'gcn',
              'dataset': 'cora',
              'dropout': 0.5,
              'epochs': 200,
              'feature': 'hww'
            }
```

Figure .9: li et al run result.

Conclusion

This is the way we execute our methods for get a real result and make a comparison.