UNIVERSITÀ
DEGLI STUDI
FIRENZE

University of Florence
Department of Information Engineering
Ph.D. Program in Smart Computing

Graph Neural Networks for Advanced Molecular Data Analysis

Candidate
Niccolò Pancino

Supervisor
Prof. Monica Bianchini

PhD Coordinator
Prof. Stefano Berretti

Cycle XXXV, years 2019-2023

Ph.D. Program in Smart Computing
Unversity of Florence, University of Pisa, University of Siena

PhD Thesis Committee:

Prof. Marco Maggini

Prof. Marcello Sanguineti

Prof. Thomas Gärtner

Prof. Paolo Nesi

Prof. Luca Oneto

Prof. Filippo Maria Bianchi

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Smart Computing.

Abstract

Graphs are data structures composed of collections of nodes and edges, which can be used to represent objects, or patterns, along with their relationships. Deep Learning techniques, and in particular Deep Neural Networks, have recently known a great development and have been employed in solving tasks of increasing complexity and variety. In particular, Graph Neural Networks (GNNs) have been extensively studied in the last decade, with many theoretical and practical innovations. Indeed, their main feature is the capability of processing graphs with minimal loss of structural information, which has caused GNNs to be applied to an increasing number of problems of different nature, leading to the development of new theories, models, and techniques. In particular, biological data proved to be a very suitable application field for GNNs, with metabolic networks, molecules, and proteins representing just few examples of data that are naturally encoded as graphs.

In this thesis, a software framework for implementing GNN models was developed and discussed. Furthermore, some applications of GNNs to molecular data, relevant both from the point of view of Deep Learning and Bioinformatics, are discussed. The main focus of the work is on the drug side–effect prediction problem. This is a challenging task, since predictions can be based on homogeneous as well as heterogeneous and complex data, with graphs collecting nodes and edges representing different entities and relationships. On the other side, protein–protein interfaces can be detected by identifying the maximum clique in a correspondence graph of protein secondary structures, a problem which can be solved with Layered Graph Neural Networks (LGNNs). Promising experimental outcomes offer valuable insights and permit drawing interesting conclusions about the abilities of GNNs in analyzing molecular data. Given the growing interest of the AI research community on graph–based models, these applications, inspired by real–world problems, constitute a very good testing ground for evaluating GNN computational ca-

pabilities, in order to improve and evolve the actual models and extend them to more complex tasks, particularly in the biological field.

Contents

List of acronyms used in the Thesis

AI : Artificial Intelligence
ANN : Artificial Neural Network
ASA : Accessible Surface Area
CGNN : Composite Graph Neural Network
CNN : Convolutional Neural Network
DL : Deep Learning
DNN : Deep Neural Network
DSE : Drug Side–Effect
GAN : Generative Adversarial Network
GAT : Graph Attention neTwork
GCN : Graph Convolution Network
GNN : Graph Neural Network
MGSEP : Molecular Graph Side–Effect Predictor
LGNN : Layered Graph Neural Network
LSTM : Long Short Term Memory
MG^2N^2 : Molecular Generative Graph Neural Network
ML : Machine Learning
MLP : Multi–Layer Perceptron
MPNN : Message Passing Neural Network
NLP : Natural Language Processing
RL : Reinforcement Learning
RNN : Recurrent Neural Network
SSE : Secondary Structure Element
SVM : Support Vector Machine
VAE : Variational AutoEncoder

Chapter 1

Introduction

Machine Learning (ML) — especially if declined in the current Deep Learning (DL) framework — is an ever–evolving field which has seen many breakthrough discoveries in recent years, allowing for solutions to be found for an increasing variety and complexity of problems. Particularly, attention is given to Graphs Neural Networks (GNNs) [1], which are a powerful tool for understanding complex relationships within graph data, and have been proven to be effective in a wide range of applications.

This thesis aims to explore the potential of GNNs in the field of biology and contribute to the ongoing research in this area.

1.1 GNNs in Bioinformatics

A graph is a data structure composed of a collection of nodes and edges which can be used to represent objects, or patterns, along with their relationships [2]. Nodes and edges can be associated with feature vectors, describing their attributes. Nowadays, graphs play an important role in many modern applications, since they are widely used to describe the information of interest in many real–world problems in different fields, including physics, social science [3], economics, and bioinformatics. For instance, for biological or chemical data, nodes denote entities, such as atoms, proteins, or genes, while edges represent chemical bonds, physical contacts, or metabolic interactions. Actually, graphs constitute the natural data domain in many bioinformatics applications, such as, for instance, molecular property prediction [4], identification of interfacing amino acids in Protein–Protein Interaction (PPI)

[5, 6], prediction of polypharmacy side effects [7], and drug design [8]. More generally, it can be observed that a graphical representation allows information from different sources to be merged in a natural way (e.g. gene regulatory networks, metabolic networks, any drugs taken) while preserving the interactions that naturally occur in a complex biological environment.

Traditional machine learning methods are not able to process graph–structured data in its native form, and require them to be encoded into vectors, with an inevitable loss of useful information and preventing models from considering the relational information inherent in the task.

GNNs are powerful connectionist models for graph–structured data, which have become practical tools for any problem involving graphs, thanks to their capability of processing relational data directly in graph form and calculating an output at each node or edge, with minimal loss of information. Indeed, GNNs assume that the input domain is represented by a set of entities and relationships between them.

Since the seminal work in [1, 9], many GNN models have been proposed, such as Graph Convolution Networks (GCN) [10, 11, 12], GraphSAGE [13], and Graph Attention Networks (GATs) [14]. GNNs have been successfully applied to a wide variety of tasks [15], spreading from computer vision [16, 17, 18] and social and recommendation systems [19, 20, 21], to biological and chemical tasks in protein–related problems [22, 4, 23, 24] and drug—discovery applications [25, 8].

Three types of tasks can be faced by means of GNNs, i.e. node–focused, edge–focused, and graph–focused problems. In particular, node–focused problems concern situations in which all the nodes of a graph, or a subset of them, are associated with a target value: intuitively, an output must be produced in correspondence of each targeted node, which can be used for classification, regression, or clustering purposes. For example, localizing a particular compound in a macromolecule, when the molecule is represented as a graph, is a node–focused task. On the other hand, edge–focused problems concern tasks in which the targets are associated to the edges: the GNN must classify, cluster, or even predict the existence of relationships between patterns. Predicting the nature of chemical bonds between atoms or molecules represents an edge–focused task. Finally, a graph–focused task concerns problems in which a unique target is associated to the whole graph, and the goal is to predict a property or to cluster the complex object represented by the graph. Predicting the mutagenicity of a particular compound

[4] is an example of graph–focused problem.

GNNs can therefore be applied to many types of problems, in both the regression and classification settings, as well as in both the inductive and mixed inductive–transductive learning framework, which makes these models extremely adaptable and capable to match the extraordinary variety of biological problems on graphs. GNNs can also be modified in order to solve even more specific tasks: they can be used to develop graph generative models [26]; attention mechanisms can make the models explainable [27]; hierarchical versions can process large graphs with complex structures [28]; composite GNNs can process heterogeneous graphs [29].

1.2 Thesis Summary

The primary objective of this thesis is to investigate the use of graph neural networks in solving biological tasks inspired by real–world problems and possibly related to molecular data. A comprehensive overview of the various ML algorithms which can be applied to structured data as well as a detailed review of the related literature in the field is provided, with a specific focus on GNNs and biological problems which can be addressed by them, in order to establish a solid foundation for the subsequent presentation and discussion of the work.

Firstly, the development of a software framework for the implementation of the original (recurrent) GNNs for a wide variety of possible scientific applications is discussed. The software has been used in all the other research works presented in this thesis. Four applications, which are relevant from the point of view of ML as well as from that of bioinformatics, are then illustrated in the following chapters.

Three applications consist in the prediction of Drug Side–Effects (DSEs) with GNNs. Predicting side–effects is a key problem in drug discovery: an efficient method for anticipating their occurrence could cut the costs of the experimentation on new drug compounds, avoiding predictable failures during clinical trials. A dataset is built for this task, which includes relevant heterogeneous information coming from multiple well known and publicly available online resources. The dataset consists of a single heterogeneous graph, where genes and drugs are represented as nodes, connected by three different sets of edges, with relationships based on gene–gene interactions, drug–gene interactions, and drug–drug similarity. Drug nodes are associ-

ated to a set of 360 common side–effects, in a multi–class multi–label node classification setting, exploiting a composite GNN model, specialized on heterogeneous graph–structured data. Since the purpose of the model is that of predicting DSEs of new drugs based on those of previously known compounds, transductive learning is exploited to better adapt the model to this setup. Given its nature, the problem is particularly challenging in the ML scope, providing an interesting application case of GNNs to a complex task (multiple non–mutually exclusive classes to be predicted in parallel), on heterogeneous data, and in a mixed inductive–transductive setting. Results have been promising in terms of accuracy compared to both other graph–based models and traditional ML methods, which are not able to use relational information, showing that encoding data in a graph structure brings an advantage over unstructured data, and that GNNs exploit the given information better than concurrent models. The method is adaptable and can be easily expanded to include more node attributes and edges without changing the ML framework. However, DruGNN operates as a black–box approach, like many other DNN–based methods: producing a much more interpretable and trustworthy version of the model in the future would boost the usability of the approach. The second application of GNNs for DSE prediction is addressed as a graph–focused multi–class and multi–label classification problem. As the drug structure can be efficiently encoded by a graph, which retains all structural information associated with the drug and which can also be enriched with relevant chemical features of the compound, GNNs are employed to predict DSEs based solely on the drug molecular graph representation. A novel dataset of molecular graphs is then introduced for the task. The experimental results show that the DSE prediction can be effectively accomplished with this method. Finally, the last application for DSE prediction includes both the setting of the previous two applications, being based on the same heterogeneous graph — including two types of nodes, representing drugs and genes, and three types of edges, for drug–gene, drug–drug, and gene–gene relationships — on which the first application is trained, and on the molecular graph representation considered in the second one. Formally, the task is still a node–focused, multi–class, multi–label classification problem: the GNN model is trained to predict the DSEs associated with the drug nodes, by processing directly graph–structured data and by generating neural fingerprints at learning time, which can be adapted to the task the model is trained for. This results in further improvements over the other

methods, showing promising performance in predicting DSEs. It is a matter of future research to specialize the predictors, e.g. considering tissue specific data and DSE supervisions, for more accurate and informative insights, as well as to provide interpretability and explainability of the models, in order to improve the usability of the approaches and to facilitate the development of safer and more effective drugs.

The last GNN application consists in the prediction of protein–protein interfaces. Simulating the structural conformation of a protein in a computer environment (in silico) can be difficult and requires a significant amount of resources. Having a reliable method for predicting interfaces could enhance the prediction of protein's quaternary structures and functions. By representing a pair of interacting peptides as graphs, a correspondence graph can be created, describing their interaction and accounting for structural similarities and contacts between the pair. The correspondence graph is then analyzed in search of the maximum clique (the largest fully–connected sub–graph within a give graph), which corresponds to the location of the interface between the two peptides. Identifying cliques in this context can be challenging due to the imbalanced distribution of data (only a small number of nodes belong to cliques) and the complex nature of the data structures. GNNs provide a viable solution for dealing with this task, with their capability of approximating functions on graphs. The experimental results show that this solution is very promising, also compared to other methods available in the literature.

The proposed applications, additionally to their inherent relevance from the point of view of research on molecular data, are a good testing ground for GNNs. The first three applications are strictly correlated, as they propose three different models in order to face the same problem. In particular, the first application demonstrates the capabilities of GNNs in processing a heterogeneous relational dataset, built from multiple different data sources; the second one is based on a homogeneous relational dataset, yet it allows to tackle the same task; finally, the third one further broadens the field of study by including both the previously described settings. This application is more classical, yet it allows to face a relevant biological task, and to test GNNs on the maximum clique detection problem (known to be NP–complete) on an unbalanced dataset. Therefore, all the presented real–world applications allow to obtain relevant results, and to draw interesting conclusions, from the point of view of ML.

Additionally, this thesis also explores other works, mainly related to bioinformatics and structured data. These studies are briefly summarized and discussed in Chapter 7. The rest of this section summarizes the contributions of the thesis, presented in Section 1.2.1, and the thesis structure, described in Section 1.2.2.

1.2.1 Main Contributions of the Thesis

The main contributions of the thesis are summarized in the following.

1. A software framework, described in the publication [**A2**], based on TensorFlow 2 and Keras, for the implementation of Recurrent GNNs in multiple application scenarios, for node–focused, edge–focused, and graph–focused applications, for homogeneous and heterogeneous graph processing, and for both inductive and mixed inductive–transductive learning settings.

2. Three GNN–based DSE predictors, described in the publications [**A1**], [**A4**], and [**A7**]:

   (a) A novel relational dataset which integrates multiple information sources into a network of drugs and genes, with different types of features and connections;

   (b) Two novel GNN–based predictors on such dataset, which deal with a multi–class multi–label classification task on heterogeneous data, both in the inductive and a mixed inductive–transductive learning framework;

   (c) Ablation studies on DSEs and data sources, which underline their importance for the learning process;

   (d) Experimental validation of the three predictors, with interesting results highlighting the gap with a non–graph–based method.

3. A GNN–based method for the identification of protein–protein interfaces, described in the publication [**P1**]:

   (a) A novel dataset of graphs describing protein–protein interactions, built on reliable and publicly available sources;

(b) A GNN predictor trained on this dataset, which successfully detects cliques (corresponding to interactions) in a highly unbalanced setting, with very promising results.

1.2.2 Structure of the Thesis

The thesis is divided into chapters, sections, and subsections to present a clear and organized structure for the research conducted during the Ph.D. program.

After the present introductory chapter, Chapter 2 describes the general concepts of DL and its application to structured data, with a focus on the GNN model and its application in Biology and Bioinformatics. A review of the literature on the topics relevant to this thesis is provided in Chapter 3, with some interesting insights on theoretical and practical applications of GNNs to biological problems, as well as on the classical methodologies and state–of–the–art approaches for the tasks addressed in this thesis.

The main contributions of this work are presented and discussed in Chapters 4, 5, and 6. In particular, Chapter 4 describes a Python–based software framework for the implementation of the original recurrent GNN model. Then, three different GNN–based DSE predictors are proposed and described in Chapter 5, along with a brief description of the publicly available data sources used for such applications. Two novel relational datasets have been built from the aforementioned databases: the former, exploited both in Section 5.3 and Section 5.5, is composed of drugs and human genes, to consider all the heterogeneous data relevant for the prediction of DSEs; the latter is a homogeneous dataset composed of molecular graphs and it is described in Section 5.4. The GNN predictors have been trained and tested on these datasets, leading to very promising results. Finally, the last contribution of this thesis is discussed in Chapter 6, in which a GNN model for the prediction of protein–protein interfaces is presented, with interesting experimental results. In this application, the GNN model is trained on a dataset composed of multiple homogeneous graphs — accounting for structural similarities and contacts between pairs of peptides — in order to detect the maximum clique on such graphs, corresponding, from a biological point of view, to the localization of the interfaces between the two peptides.

Other works, carried out during the Ph.D., not related to the main contributions of the present work, yet still relevant to ML and bioinformatics,

are briefly described in Chapter 7.

Lastly, Chapter 8 draws the conclusions of this thesis, suggesting possible future developments, and discussing their relevance and meaning.

Chapter 2

Deep Learning on Structured Data

This chapter introduces the fundamentals of deep learning (DL) and discusses its application to structured data, specifically graphs. It also provides an overview of the types of biological data that are particularly well suited for DL approaches and introduces the main applications that are discussed in the thesis. In particular, Section 2.1 describes DL in the wider framework of Artificial Intelligence (AI) and Machine Learning (ML), briefly introducing it from a historical point of view in Section 2.1.1, along with some insight on deep neural networks and on *ad hoc* learning algorithms (Section 2.1.2). Models and algorithms for structured data are introduced in Section 2.2. In Section 2.3, Graph Neural Networks (GNNs) are introduced, sketching the general model in Section 2.3.1, its learning procedure in Section 2.3.2, and its deeper version represented by the Layered Graph Neural Networks (LGNNs) in Section 2.3.3, as well as the composite model for heterogeneous graphs in Section 2.3.4. Moreover, a theoretical analysis of the approximation capabilities of GNNs and an overview of the principal models and applications are given in Sections 2.3.5 and 2.3.6, respectively. Finally, Section 2.4 describes how GNNs are applied to molecular data, reviewing biological problems involving graph data in Section 2.4.1, giving a deeper view on graph–based drug related tasks in Section 2.4.2, and introducing the applications of GNNs to some bioinformatics problems in Section 2.4.3.

2.1 Deep Learning

DL is a subset of a larger family of ML methods based on ingesting data representations with deep architectures, i.e. ANNs with three or more layers, to solve complex tasks: while a neural network with a single layer can still make approximate predictions, additional hidden layers can help to refine the knowledge, extracting more and more abstract and high level information from data. These neural networks attempt to simulate the behavior of the human brain, learning from large amounts of data, ranging from systems that simply replicate solutions designed by human experts, to ML models that learn their solutions from experience.

DL architectures, for example, have been applied in computer vision, automatic spoken language recognition, natural language processing, audio recognition, and in bioinformatics, which is the science that studies the use of computer tools to analyze biological phenomena such as gene interactions, protein composition and structure, and biochemical processes in cells.

From a mathematical point of view, a ML model learns a function f associating an output y to an input x, according to its parameters θ, as described in Eq. (2.1):

$$f(x, \theta) = y \tag{2.1}$$

The ML and DL models are also capable of performing different types of learning, which are usually classified as supervised learning, unsupervised learning, and reinforcement learning. Supervised learning utilizes labeled datasets, where the correct value $\hat{y}$ of $f(x)$ is known, to categorize or make predictions; this requires some kind of human intervention to label input data correctly. In contrast, unsupervised learning does not require labeled datasets, and instead, it detects patterns in the data, clustering them by any distinguishing characteristics. Therefore, unsupervised models learn from unlabeled data, using only the available information on examples. Reinforcement learning is a process in which a model learns to become more accurate for performing an action in an environment based on feedback, in order to maximize the reward.

2.1.1 From Artificial Intelligence to Deep Learning

The history of AI can be traced back to 1943, when Walter Pitts and Warren McCulloch created a neuron model inspired by the neural cells of the human

brain [30], with a combination of algorithms and mathematics to mimic the thought process.

In 1950, Alan Turing wrote a paper [31] suggesting how to test a *thinking* machine: he believed a machine could be described as thinking if it could carry on a conversation imitating a human with no noticeable differences, by way of a teleprinter. His paper was followed, in 1952, by the Hodgkin–Huxley model [32] of the brain as composed by neurons forming an electrical network, with individual neurons firing on/off pulses. These combined events, discussed at a conference sponsored by Dartmouth College in 1956, helped to spark the idea of AI.

The concept of ML was first introduced in a 1959 study [33], which can be considered one of the first works on learning algorithms, although it contained only basic concepts and ideas, without proposing a learning mechanism usable and efficient.

AI has had a mostly steady evolution with two substantial interruptions known as the *AI winters*. The first AI winter occurred from 1974 to 1980, as the US government halted financing for AI research. AI researchers faced two very basic obstacles: not enough memory and processing speed which would be considered appalling by modern standards. With the promising debut of "Expert Systems" (ES), which were created and rapidly adopted by leading competitive organizations around the world, the first AI winter came to a close. The issue of gathering information from multiple specialists and disseminating it to its consumers was the main focus of AI research.

Anyway, the real breakthrough did not come before the 1980s, with the introduction of the BackPropagation algorithm [34], i.e. the essence of neural network training, based on the backward propagation of errors for tuning the parameters, reducing error rates and making the model reliable by increasing its generalization capabilities.

The AI industry experienced a severe decline in research from 1987 to 1993, which coincided with the increasing popularity of desktop PCs and the perception that expert systems were too expensive and difficult to maintain. DARPA also redirected its funding to other initiatives, leading to a reduction in funding for AI research and the Second AI Winter began. However, some individuals continued to work on machine learning, resulting in significant progress, such as the development of the support vector machine model in 1995 [35] and the LSTM (Long Short–term Memory) recurrent networks in 1997 [36]. Ironically, AI thrived in the absence of government funding and

public outcry.

Overall, many of AI's historic goals were achieved in the 1990s and 2000s: Grandmaster and then world chess champion Gary Kasparov lost to IBM's Deep Blue, a chess–playing computer program, in 1997. Kasparov's defeat had worldwide resonance and represented a sensational boost for the development of AI.

ML made significant progress when GPUs were created in 1999, allowing for faster processing of data and the development of deep learning algorithms. Over a ten–year period, faster processors with GPUs increased computational speed 1,000 times. Neural networks became a contender against support vector machines, with superior results over time as more training data was provided, resulting in a collection of algorithms to model data high–level abstractions through the use of deep architectures, spreading the concept of DL and Deep Neural Networks (DNNs).

Before DL could know the fast and revolutionary development it has known in the 2010s (which is still ongoing at the present day), the issue to be solved were long–term dependencies in data, and the consequent vanishing gradient problem [37]. Many different solutions were proposed, with the development of models based on different paradigms, and designed for different data, that will be overviewed in Section 2.2.1.

ML includes many different paradigms of learning machines:

- RL [38] investigates how agents can be taught to operate so that to solve a problem by means of rewards (reinforcements);
- Support Vector Machines (SVMs) are trained to discriminate data linearly, or with kernels accounting for non–linearity [35];
- ANNs combine artificial neurons [30] to approximate non–linear functions of variable complexity;
- Self–Organizing Maps (SOMs) are a special type of ANN, based on competitive rather than inductive learning; they are an unsupervised model: the neurons are arranged into a regular grid (map) and are able to fire based on the input and the activation of their neighbors [39];
- Clustering methods are unsupervised learning algorithms that can be trained to associate data entities based on their vicinity in the feature space [40];

- Random Forests (RFs) [41] learn to build ensembles of decision trees that fit training data according to supervisions;
- Gradient boosting techniques [42] realize a strong decision model by building an ensemble of several weak decision models (usually decision trees).

A particular family of machine learning models, known as ANNs, has gained a lot of popularity in recent years for addressing problems of increasing complexity, frequently surpassing other ML techniques. ANNs use artificial neurons, the basic processing units which apply a function to the weighted sum of its inputs [30]. ANNs are commonly organized into layers of neurons, and their architecture determines their ability to resolve problems of varying complexity. More layers and units per layer improve ANN computing power, but training becomes more challenging due to theoretical and practical constraints such as memory requirements and processing speed. The number of layers defines the depth of the network: DNNs refer to networks with multiple hidden layers, e.g. layers which are not externally observable and are located between the input and the output layer. The more layers a neural network has, the "deeper" it is. A simple type of ANN is the Multi–Layer Perceptron (MLP), which takes an array of features as input and computes an output function defined according to the problem at hand. MLPs have been proven to be universal approximators for Euclidean data [43, 44, 45]: this means that the mapping between input and output vectors can be learned by an MLP, with a sufficient number of neurons [44] and at least one hidden layer, with any degree of accuracy. The first example of a DNN can be represented by the four–layer Cognitron [46]. More complex ANNs and DNNs have been developed to handle structured data and are considered in Section 2.2.

2.1.2 Learning with Deep Models

Feedforward models are the simplest type of ANNs, which consist of multiple layers of computational units, through which the information flows unidirectionally, from the input to the output layer. Each neuron in one layer has directed connections to the neurons of the subsequent layer. More specifically, in this network, the input signal is propagated through the N layers $L_1, L_2, ...L_N$ in only one direction — forward — from the input nodes,

through the hidden nodes (if any) and to the output nodes, obtaining the network output y. There are no cycles or loops in the network. In a supervised learning setting, the output y is then compared to the supervision $\hat{y}$, by means of an error function $E(y, \hat{y})$, also called *loss function* or cost function. The model is optimized with a gradient descent algorithm in order to minimize E: the process starts from the calculation of the derivative $\partial E/\partial y_i$ for each unit i of the output layer L_N, which can then be used to calculate the contribution to the error of the units belonging to the last hidden layer $\partial E/\partial y_j$, for each unit j of the hidden layer L_{N-1}, as in Eq. (2.2), and so on:

$$\frac{\partial E}{\partial x_j} = \sum_{i \in L_N} \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial x_j} \tag{2.2}$$

In fact, the process is repeated in cascade — as a backward calculation of the error contributions — and is therefore called BackPropagation [34]. These contributions are then exploited to learn a better configuration and therefore to update the network parameters θ. DNNs, where the number of layers N is large, are characterized by the long–term dependency problem [37], which prevent the model to learn the dependencies between neurons located in distant layers due to the so called vanishing gradient problem: the derivative of the error gets so small when backpropagating to lower network layers, that the layers closer to the input cannot be trained from experience. The source of the problem turned out to be the use of certain activation functions, which condensed their input, in turn reducing the output range in a somewhat chaotic fashion. This produced large areas of input to be mapped over an extremely small range. In these areas of the input space, a large change will be reduced to a small change in the output space, resulting in a vanishing gradient. This behaviour prevents traditional ML algorithms, like standard BackPropagation, from successfully training DNN models [37].

Over the years, in order to solve these issues, new ad–hoc training methods, activation functions, such as the Rectified Linear Unit (ReLU), and batch normalization, were introduced.

2.2 Machine Learning for Structured Data

Structured data are everywhere and they are playing an increasingly important role in our daily lives as the world becomes more interconnected. As

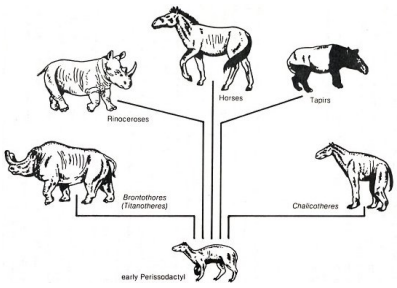
the amount and variety of data related to any given problem increases, the use of relational data and data structures becomes vital. Indeed, structured data are particularly useful in organizing and managing large amounts of information, allowing for more efficient analysis and decision–making. In a variety of fields, such as finance, healthcare, and transportation, structured data are essential for optimizing processes and achieving better outcomes. As our reliance on data continues to grow, the importance of structured data will only continue to increase [15].

In the field of AI and ML, data are often organized and processed in a particular way. The most common data type, called Euclidean, consists of entities described by a simple vector of feature values. This type of data is often used by traditional ML models and ANNs. However, real–world data can be represented with many different structures, including sequences, trees, graphs, and images. A sequence is a type of data structure that represents each entity as part of a temporal or spatial succession of similar entities, where each entity is followed (and preceded) by only one other entity: examples of data that can be represented as sequences include nucleic acids, proteins, temporal sequences of weather observations, and item queues. Trees are a generalization of sequences, where each entity can be followed by multiple entities (but preceded by only one *parent* entity), such as phylogenetic trees, decision trees and, in general, data admitting a hierarchical organization. Graphs are a further generalization of trees, where entities are represented as nodes and relationships between entities are represented as edges. Graph data examples can include protein structures, metabolic networks, molecules, power grids, traffic systems, citation networks, knowledge graphs, social networks, and even the Internet. Images are a specific type of graph, where pixels are represented as nodes and edges only exist between nearby pixels. Images can represent a wide variety of subjects, and can be used for helping with the identification of tumors through radiography and skin images, analyzing road usage through images of vehicles, or detecting emotions through photos of human faces. Often, combinations and hierarchies of these structured data types can be found. For instance, videos are sequences of images, networks of interacting compounds can be seen as graphs of graphs, phylogenetic trees based on DNA are trees of sequences, and the weather can be analyzed using sequences of graphs. Visual examples of these structured data types can be found in Fig. 2.1.

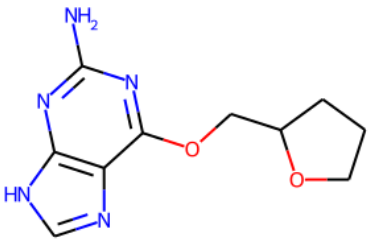
Traditionally, these structures are encoded into Euclidean feature vectors

using specialized algorithms: however, this process can result in a loss of information. Deep learning solutions, on the other hand, can analyze and make decisions based on structured data in its natural form, potentially improving the accuracy and effectiveness of the analysis. Such solutions will be analyzed in Section 2.2.1.

(a) Sequence



(b) Tree



(c) Graph

(d) Image

Figure 2.1: Some structured data examples. (A) Sequences: The two strands of a DNA double helix. (B) Tree: a cladogram. (C) Graph: the structural formula of a molecule. (D) Image: a photo is a collection of numerical values of pixel colour levels.

2.2.1 Structure–Oriented Models

Deep neural networks are a type of artificial neural networks which can be designed with complex architectures to better fit the structure of the data they are intended to process. One of the earliest approaches for training neural networks with BackPropagation, the BackPropagation Through Time (BPTT) algorithm, was published in 1990 [47], just a few years after the introduction of BackPropagation for feedforward neural networks [34]. One of

the first types of structured data that was processed with artificial neural networks — namely Recurrent Neural Networks (RNNs) — using the BPTT algorithm was sequences [48]. RNNs use the concept of recursion by replicating the same layer (or group of layers) for each element in an input sequence. In 1997, the introduction of LSTM networks [49] revolutionized the field by introducing the concept of cell gates, which are units that can control the flow of signals (and gradients) through the network, allowing it to store information over an extended period of time [49]. Gated recurrent units (GRUs) are similar to LSTMs, but they only have a single gate per unit (the forget gate) [50]. RNNs have been used for tasks such as natural language processing [51], protein secondary structure prediction [52, 53], motion recognition [54], stock market prediction [55], and speech recognition [56]. These same application areas have also been explored using one–dimensional Convolutional Neural Networks (CNN–1D) [57] and transformers [58]. Transformers have had a significant impact on the field by introducing an attention mechanism that allows the network to weigh the importance of different elements in a sequence and overcome biases introduced by the order of the elements [58].

Images are traditionally more complex to process, since they can be large in term of size and memory consumption. Moreover, learning on images with traditional ML models such as MLPs is not efficient, since slicing them to fit into vectors can lead to structural information loss, as well as processing them can be computationally expensive or even infeasible. Convolutional neural networks were developed as a solution to this problem. Based on a theory from 1989 [59] and first introduced in 2012 [60], CNNs use small patches of pixels as inputs and produce a single output value, repeated across the whole image, without any need to slice the image. This process can be repeated with different convolutional filters in the same layer, or with multiple convolutional layers. CNNs also often include pooling layers, to reduce the input size, and dense layers. There are many different types of CNN architectures that have been developed for a range of tasks, including image classification [60], segmentation [61], object detection [62], and image generation [63], introducing deeper models as the research goes on [64]. These models change the paradigm of forward propagation: each residual block (composed of a small number of layers) learns to refine the output of the previous residual block, shortening the gradient path in BackPropagation [65]. CNNs can be very deep, with hundreds of layers, and often use resid-

ual connections between layers to improve their performance. They can also be used to process 1D sequences, 3D images, and videos, by generating a recurrent CNN. In this regard, 1D, 2D and 3D grids can be considered as particular kinds of graphs, yet CNNs cannot be used to process any type of graph. Processing and learning on graphs are challenging tasks in ML, and various methods have been developed to extract Euclidean information from them, including techniques based on random walks [66] and kernels which can approximate functions on graphs [67], with many applications in bioinformatics [68]. Some specific types of graphs, like trees and directed acyclic graphs, can be processed using models from the recurrent neural network family [69, 70]. In this context, the breakthrough discovery was the theorization of neural networks which can process graphs by adapting their architecture to the input graph topology, namely GNNs [1]. These models will be extensively described and discussed in the following Section 2.3.

2.3 Graph Neural Networks

Graph Neural Networks (GNNs) are a well–known class of machine learning models for graph–structured data processing. The first theorization of GNNs dates back to 2005 [71] — with the full mathematical formulation proposed in 2009 [1] — and describe them as networks which replicate the topology of the input graph, and exchange messages between neighbor nodes to produce an output on selected data. GNNs typically work on a graph dataset D composed of one or more graphs, processed independently.

A graph is a non–linear (non–Euclidean) data structure composed of a collection of nodes and edges. Formally, it is defined as a tuple $G = (N, E)$, where N is the set of nodes and $E \subseteq N \times N$ is the set of edges. Nodes represent objects or entities, and relationships between them are represented by edges. Nodes and edges can be associated with values or vectors of values, describing their attributes, and defined as node feature vectors, $l_n,\ \forall n \in N$, and an edge feature vectors, $e_{n,m},\ \forall (m, n) \in E$, respectively. Moreover, a neighborhood function Ne, which associates each node n to the set of its neighbor nodes $Ne(n) \subset N$ can be defined, based on the edges E.

2.3.1 The Graph Neural Network Model

A GNN assigns a state s_n to each node $n \in N$ and updates it iteratively by sending messages through the edges connecting n to its neighbors $Ne(n)$. The GNN theoretical model is fully described by two parametric functions, f_w and g_w, which, respectively, regulate the state updating process and the output calculation.

The GNN model approximates an output function g_w, expressing a property of the whole graph G, of its nodes or a subset of its nodes $N_{out} \subseteq N$, or of its edges or a subset of its edges $E_{out} \subseteq E$. To do so, a state $x_n \in \mathbb{R}^{d_x}$ is associated to each node $n \in N$, and then iteratively updated by sending messages through the edges connecting n to its neighbors $Ne(n)$.

The state dimension d_x as well as the number of state updating iterations K are hyperparameters of the GNN, while the states are usually initialized by sampling from a random distribution centered on the origin of $\mathbb{R}^{d_x}$. Given the randomly sampled initial states $x_n^0,\ \forall n \in N$, the state of a generic node n at iteration t can be calculated as in Eq. (2.3):

$$x_n^t = f_w(x_n^{t-1},\ l_n,\ A(\{(M(n,\ m,\ t)) : m \in Ne(n)\})) \tag{2.3}$$

where $M(\cdot)$ and $A(\cdot)$ are the the function defining the message coming from the neighbourhood $m \in Ne(n)$ to node n, and the aggregating function defining how these messages are aggregated, respectively. M can be any function returning the message in the form of a vector whose computation is based on the destination node n, the source node m, and the label $e_{m,n}$ of the edge (m, n) connecting the two nodes. M could even be learned with a neural network. In this thesis, M always takes the general form defined in Eq. (2.4) with the possibility of excluding l_m or $e_{m,n}$ (when nodes/edges are not labeled) from the computation:

$$M(n,\ m,\ t) = (x_m^{t-1},\ l_m,\ e_{m,n}) \tag{2.4}$$

On the other hand, the aggregation function A can be any function defined on a set of vectors (the messages), each having dimension d_m and returning a single vector of the same dimension. A is usually the sum, average, or maximum of the single components of the message, but it could even be learned with another neural network, as for example in the (convolutional) aggregations of GraphSAGE [13]. In this thesis, A is either the

sum or average of the messages, as described by Eq. (2.5):

$$\begin{aligned} A_{sum} &= \sum_{m \in Ne(n)} (x_m^{t-1},\ l_m,\ e_{m,n}) \\ A_{avg} &= \frac{A_{sum}}{|Ne(n)|} \end{aligned} \tag{2.5}$$

Since the two aggregations are similar, a hyperparameter a equal to $1/|Ne(v_i)|$ or 1 can be defined to select the aggregation function, obtaining the average or the sum, respectively. Combining all these concepts, the state updating function in its general final form can be rewritten as in Eq. (2.6):

$$x_n^t = f_w(x_n^{t-1},\ l_n,\ a \sum_{m \in Ne(n)} (x_m^{t-1},\ l_m,\ e_{m,n})) \tag{2.6}$$

The GNN implements a recurrent algorithm for exchanging useful information between nodes and their neighbors, for a pre–set number of iterations T or until the state computation dynamics reaches a stable equilibrium point at the iteration $t \leq T$; then, the final versions of the node states $x_n^T,\ \forall n \in N$, are fed in input to the output network, approximating the output function g_w, which can be defined for the three types of problems addressed by GNNs: node focused, edge focused or graph focused. In node focused problems, the output is defined for each node $n \in N_{out}$, as a function of its state and label, as in Eq. (2.7):

$$y_n = g_w(x_n^T,\ l_n) \tag{2.7}$$

In edge focused problems, the output is defined for each edge $(m, n) \in E_{out}$, as a function of the states of both the source node m and the destination node n, and, if it exists, the label $e_{m,n}$, as in Eq. (2.8):

$$y_{m,n} = g_w(x_n^T,\ x_m^T,\ e_{m,n}) \tag{2.8}$$

Finally, in graph focused problems, the output is calculated over each node $n \in N_{out}$ as in node based problems, and then averaged over the output nodes. This is defined in Eq. (2.9):

$$y_G = \frac{1}{|N_{out}|} \sum_{n \in N_{out}} g_w(x_n^T,\ l_n) \tag{2.9}$$

2.3.2 Learning with Graph Neural Networks

Based on an information diffusion mechanism, GNNs can process homogeneous and heterogeneous graph–structured data. In particular, GNNs create an *encoding network*, a recurrent neural network architecture which replicates the topology of the input graph, by means of two MLP units, one implementing the state transition function f_w of Eq. (2.6) at each node and the other implementing the output function g_w (on targeted nodes or edges). The GNN replicates the MLP units on each node of the input graph and unfolds itself in time and space, generating a feedforward architecture, known as the *unfolding network*, in which each layer contains copies of all the elements of the encoding network and represents an iteration of the implemented algorithm. Connections between neurons belonging to subsequent layers reproduce exactly those of the encoding network. Weight sharing is exploited between all the copies of the MLPs, allowing to manage long–term dependencies between distant nodes in the graph, for both the state and output MLPs. The information associated with each node can thus be propagated through the whole graph in a sufficient number of iterations; then an output on nodes, edges, or whole graphs — depending on the problem under analysis — is produced by the MLP implementing g_w.

A sketch of the unfolded encoding network is provided in Fig. 2.2.

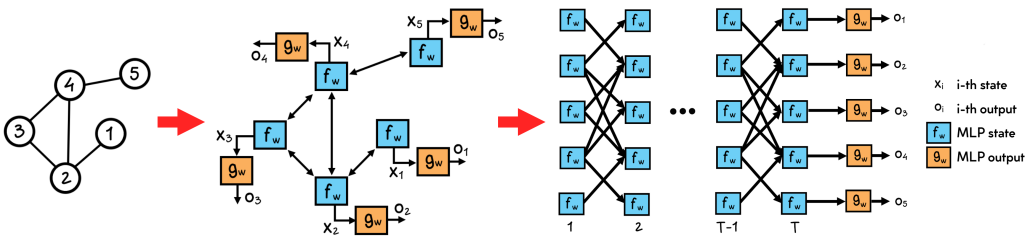


Figure 2.2: Sketch of how the GNN model produces the encoding network and its unfolded version on an example input graph (for a node focused problem). In the unfolded network, the topology–like replica of the input graph is guaranteed by the connections between layers.

Note that the unfolded encoding network corresponds to a DNN with recurrent layers: given the numbers of layers in the state MLP, indicated with L_f, and in the output MLP, denoted with L_g, the network has a depth of $K \times L_f + L_g$ layers. Weight sharing in space and time makes the model scalable, invariant in the number of parameters to graph size and number of iterations, and less prone to overfitting. Consequently, to optimize the

parameters of the network, and therefore implement the learning process, it is sufficient to apply a standard optimization algorithm based on Back-Propagation. Typical examples include stochastic gradient descent or the Adam optimizer [72]. A loss function, depending on the problem under analysis (e.g. cross–entropy for classification), is applied to the outputs and the targets, computing the error. The error gradient is then calculated with respect to all parameters of the network, averaging the contribution over all the replicas of each parameter, and applying the resulting value to all the replicas of the same weight.

2.3.3 Layered Graph Neural Networks

In 2010, the base GNN model was further refined by introducing its deeper version, the Layered Graph Neural Networks (LGNNs) [73], to face the Long–Term Dependency problem, which corresponds to the inability of the network to correctly process a complex structure, due to the local nature of the GNN training process, which does not allow state computation to be influenced by labels and states of distant nodes.

The LGNN architecture is characterized by multiple GNNs connected in cascade, in which each layer can be trained separately using always the same target, and using invariably the original graph as input, though with node labels enriched by the results calculated in the previous layer. Formally, the first layer is a standard GNN operating on the original input graphs. Each layer after the first is trained on an enriched version of the graphs, in which the information obtained from the previous layer is concatenated to the original node labels. This additional information consists in, either, the node state, the node output, or both. Formally, the label of node n in the i–th layer, $i > 1$, is $l_n^i = [l_n, x_n^{i-1}]$ or $l_n^i = [l_n, o_n^{i-1}]$ or $l_n^i = [l_n, x_n^{i-1}, o_n^{i-1}]$, where x_n^{i-1}, o_n^{i-1} are, respectively, the state and the output of node n at layer $i-1$ of the cascaded architecture. Given the operation described for the labels, all the mathematical formulations remain valid for each layer. The output of a node in a GNN, y_n, is influenced by the node state x_n, which in turn is influenced by the node's neighbors $Ne(n)$: if this information is included in the training data for each GNN layer, then the model is able to expand the node's neighborhood, layer by layer.
This means that the LGNN can progressively widen the neighborhood of a generic node, continually increasing the amount of information it takes

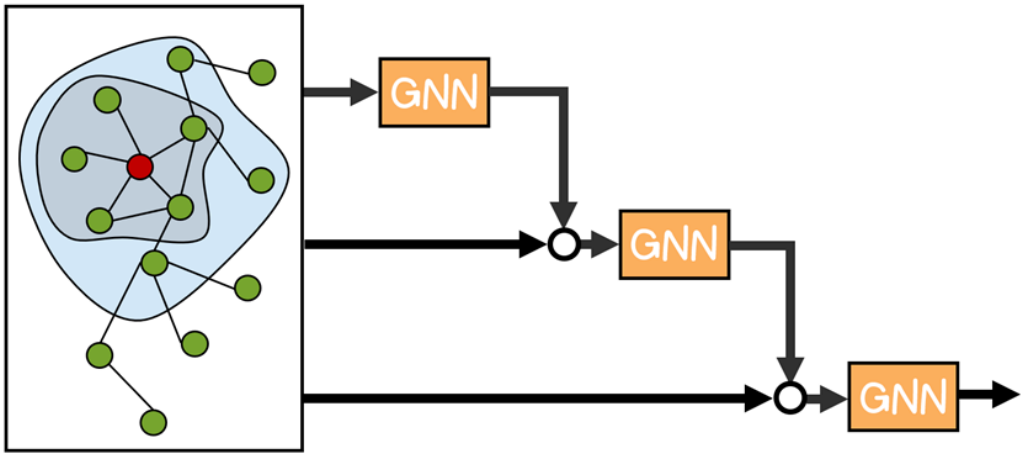


Figure 2.3: Example of an LGNN with 3 layers.

into account from nodes which are further and further away from the one under analysis. Moreover, the solution proposed by the previous layer (in the form of states or output vector, or both of them) is integrated to the input of each layer after the first, significantly addressing the long–term dependency issue: each GNN, therefore, becomes an expert which solves the considered problem using the original data and the experience obtained from the GNNs of the previous layers so as to "correct" the errors made by the previous networks, rather than solving the whole problem. As a result, LGNNs progressively refine the model's output and obtain a greater learning capability with respect to a single–layered GNN [73].

The architecture of a LGNN model applied to a generic graph is shown in Fig. 2.3.

2.3.4 Composite Graph Neural Networks

GNNs and LGNNs can be extended to the domain of heterogeneous graphs, where multiple types of nodes coexist, as they represent different kinds of objects and may have different numbers and types of features: the GNN model for heterogeneous graphs [29] is known as a Composite Graph Neural Network (CGNN), while the LGNN model in this case is called Composite Layered Graph Neural Networks (CLGNN). Heterogeneous graphs are often used to represent information about different types of entities and/or relationships: typical examples of this setting are knowledge graphs, in which entities of different types need to be encoded into a single relational graph,

or molecular graphs, since atom species can be distinguished and represented by different node types.

Without loss of generality, in this section, only the case of CGNNs will be analyzed. CGNNs process graphs with edges representing different types of relationships considering a one–hot encoding of the relationship type as a label, possibly to be concatenated to the edge features, if any. Node types instead are treated as subsets of the node set N, and each type has a dedicated state network. In this setting, given the number of node types nt in the dataset, the CGNN learning process is based on multiple MLPs for the nodes state updating process, one for each type of nodes, and a variable number of output networks, depending on the problem under analysis. Each state network F_i learns its own version $f_{w,i}$ of the state updating function in Eq. (2.6). In this way, each node type is associated with a unique MLP which learns a unique state transition function. All the F_i are characterized by the same output dimension, corresponding to the CGNN hyperparameter d_x. To allow nodes belonging to different types to exchange coherent messages, the node label is not considered as a part of the message, as it can assume different dimensions and meaning for different node types. The state updating function $f_{w,i}$ can therefore be rewritten as in Eq. (2.10):

$$x_n^t = f_{w,i}(x_n^{t-1},\ l_n,\ a \sum_{m \in Ne(n)} (x_m^{t-1},\ e_{m,n}))\ :\ i = T[n] \tag{2.10}$$

where, to select the correct $f_{w,i}$, a vector T associating each node $n \in N$ to its type i is given to the GNN as part of the dataset.

The output functions described in Eq. (2.7), (2.8), and (2.9) are still valid, with the only difference represented by the number of output network used for calculating the output: in the most general case, for node focused applications in which N_{out} contains nodes of all the types defined in the dataset, the CGNN model requires nt output networks; in the simple case, in which N_{out} refers to a single node type — as in all the applications described in this thesis —, the model still requires nt different state networks, but only a single output network is needed, since the label of only one node type is considered in calculating the output for each $n \in N_{out}$. As a consequence, in the latter case the parameters the GNN has to learn are distributed in $nt + 1$ MLPs, in contrast to the two MLPs of the homogeneous setting.

The rest of the learning process is exactly the same as the homogeneous case, as explained in Section 2.3.2, with the only difference consisting in the

number of state networks — and possibly of output networks — to build the encoding network and consequently the unfolding network. The CGNN learning process scheme on a heterogeneous graph is depicted in Fig. 2.4

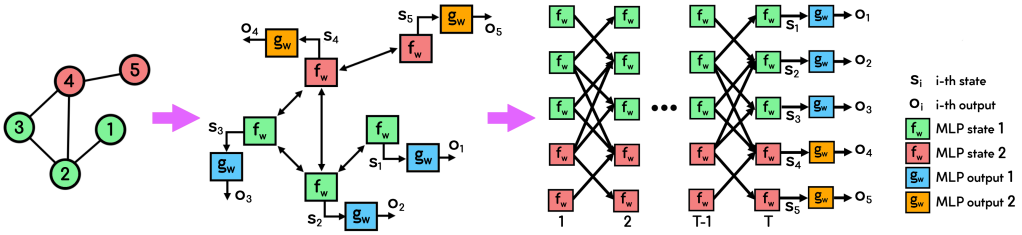


Figure 2.4: Scheme of a CGNN learning process on a heterogeneous graph. From the left: The heterogeneous input graph, where green and red circles represent two types of nodes, and connections belong to a single undirected edge type. The encoding network, generated by means of the MLPs implementing the state transition functions $f_{w,i}$ (red and green rectangles, one for each node type) and the output functions $g_{w,i}$ (blue and orange rectangles). The unfolding network, in which the information can flow from the input to the output, passing through the aforementioned MLPs and the graph connections defining this feedforward network. Note that for a non–composite GNN, learning on a homogeneous graph, the scheme is the same, with a unique MLP for state transition, since only one type of nodes is admitted.

2.3.5 Approximation Power of GNNs

The approximation capabilities of DNNs, and in particular MLPs, have been studied and described in various theoretical works [43, 44, 74]. However, since GNNs operate on the graph domain, they have to withstand different challenges, also concerning symmetries in the data structures. In [9], published in parallel with the original model [1], the computational power of GNNs was analyzed.

Given a graph $G = (N, E)$, to analyze the computation of the state of a generic node n in the graph, the concept of unfolding tree is introduced: the unfolding tree is built by taking n as the root, adding its neighbors as child nodes, and then recursively adding the neighbors of each child node as its children. In other words, an unfolding tree is obtained by unfolding G up to an arbitrary depth k using n as the starting point, as described in Eq. (2.11):

$$T_n^k = \begin{cases} \text{Tree}(x_n,\ \{T_m^{k-1}, \forall m \in Ne(n)\}) & \text{if } k > 1 \\ \text{Tree}(x_n) & \text{if } k = 1 \end{cases} \tag{2.11}$$

The unfolding tree T_n^k represents all the information on node n available to the GNN after k iterations. For k larger or equal to the diameter of G, two nodes n and m with identical unfolding trees $T_n^k = T_m^k$ are indistinguishable to the network and are said to be unfolding equivalent. It was demonstrated that GNNs are universal approximators for all functions that preserve the unfolding equivalence (i.e. functions that produce the same result on any pair of unfolding equivalent nodes) [9]. An alternative way to assess the computational capabilities of GNN models has been presented in recent publications [75, 76]. It is based on the Weisfeiler–Lehman graph isomorphism test [77], which associates a unique, canonical representation to each graph, based on the connections between nodes (i.e. the node adjacency). If two graphs have the same representation, they are considered isomorphic, meaning that they have the same structure. However, the one–dimensional version of the Weisfeiler–Lehman test (1–WL) cannot distinguish between all possible pairs of graphs, because the same representation can be shared by multiple non–isomorphic graphs. Therefore, higher–dimensional versions of the test, such as the D–dimensional test (D–WL), have been developed. These tests are based on sets of D nodes and are more effective at distinguishing between non–isomorphic graphs. The ability to distinguish between non–isomorphic graphs increases with D.

GNN models can be classified based on their ability to mimic the Weisfeiler–Lehman test: those which can replicate the 1–WL test are classified as 1–WL models, which means they are at least as powerful as the one–dimensional Weisfeiler–Lehman test; however, many currently used GNN models are not as powerful as 1–WL, because they are unable to simulate the test. Interestingly, the 1–WL test is analogous to an iteration of neighborhood aggregation in recurrent GNNs: as a result, these models have been demonstrated to be all 1–WL if injective neighborhood aggregation and output functions [75] are used. Currently, no GNN model has been able to simulate higher–order Weisfeiler–Lehman tests like, 2–WL or more [75], although some research has been done on developing higher–order GNNs using non–local neighborhood aggregation [78]. Moreover, it has been demonstrated that both unfolding trees and the Weisfeiler–Lehman test can be used to evaluate the approximation power of GNNs [79].

2.3.6 Applications of Graph–based Models

Since the seminal work [1, 9], many GNN models have been introduced [80], which can be classified, depending on how the information is propagated among nodes, in two broad families: recurrent GNNs, and convolutional GNNs. The former, which also include the original model [1], are based on message passing between nodes and apply the same weight matrices in the recurrent calculation of node states, whereas the latter apply different weights at each time–step t. Moreover, Convolutional GNNs, also known as Graph Convolution Networks (GCNs), are based on the concept of graph convolution: similarly to what happens in CNNs, a convolutional filter is applied on each node (and its neighborhood) to calculate its label in the subsequent layer, or its output.

One key difference between standard convolutional layers and graph convolutional layers is the way they handle spatial relationships between data points. In standard convolutional layers, the spatial relationships between data points are fixed and predefined by the grid structure of the data. In graph convolutional layers, on the other hand, the spatial relationships between data points are dynamic and can be learned from the data itself, since they are defined by the edges of the graph. This makes graph convolutional layers well–suited for tasks where the spatial relationships between data points are complex or not well–defined.

Examples of recurrent GNNs are Graph Nets [81], Gated Graph Sequence Neural Networks [82], Message Passing Neural Networks [83], and Graph Isomorphism Networks [75]. The first convolutional GNNs to be introduced were standard GCNs [10], followed by spectral convolution models [11, 12]. GraphSage generalized the concept of convolutional GNN by introducing different types of neighbor aggregation [13]. GCNs were also combined with attention mechanisms in Graph Attention Networks [14], improving the predictions with information on important relationships [84] and dealing with explainability issues [27].

Graph–based models can be applied on any type of graph, both in real–world applications and synthetic problems. For example, in the Web domain, GNNs have been used for spam detection [85], community detection [10], content interaction prediction [86] and sentiment analysis [3]. They have been employed also in prediction of logical relations in knowledge graphs and future links in citation networks [87], and as recommendation systems

[88, 89]. Many node or graph classification and regression tasks have been solved by applying GNNs [15], also in a heterogeneous setting [90]. Models of the GNN family have showed performance at or close to the state–of–the–art in problems of graph matching [91], weather forecasting [92], power grid analysis [93], and many others. In the biological domain, GNNs can calculate molecule properties [83, 94], predict protein–protein interfaces [23], and classify compounds according to their mutagenicity [4] or activity against HIV [95, 22], just to list a few applications.

2.4 Biological Problems on Graphs

Computational methods have enabled new research avenues in biology and medicine, resulting in the development of interdisciplinary fields such as bioinformatics and medical informatics. DL techniques are becoming more widely used in these fields, offering innovative solutions to previously untreatable problems, reducing the expenses and duration of traditional methods, and improving existing processes by making them more efficient.

2.4.1 Graph Data in Biology

Biological data are often naturally represented with graphs. Molecules have always been perceived as graphs, in which atoms correspond to nodes (possibly labelled with some chemical and physical information such as atom type, molecular weight and polar surface area), and chemical bonds correspond to edges, whose label can be the type of bond. These structures are also known in the literature as molecular graphs. Molecular graphs allow to predict many properties of each molecule, such as mutagenicity, anti–HIV or anti–cancer action, and other levels of activity. Other structures like polymers, proteins, and nucleic acids can be effectively represented as graphs, with nodes representing different structural levels (e.g. substructures, protein secondary structures, DNA blocks) and edges representing interactions between these components. This type of representation is useful for understanding and solving key biological problems, such as predicting interactions between proteins and ligands, proteins and proteins, and proteins and nucleic acids. Additionally, a hierarchical model can be used, in which each node is expanded into its own graph substructure, such as, for example, when a graph containing nodes that describe interacting proteins is expanded by

representing each protein as a molecule — in this case it is referred to as graph of graphs.

Graphs can also be used to represent logical information, such as knowledge graphs of biological entities. In these graphs, each node represents an entity and each edge represents a relationship between two entities. ML models can then be trained to predict new, biologically relevant relationships between entities based on the known relationships: this is often done in a heterogeneous setting, where the entities come from different types of data sources.

2.4.2 Graphs in Drug Discovery

Drug discovery is the discipline which focuses on how to develop new drug compounds. Discovering a new molecule is a long and expensive process [96], often involving researchers, companies, and national agencies [97]. Drug discovery is a field in constant development which is increasingly influenced by the use of ML techniques [25], and GNNs in particular [8], since drugs and other molecules are efficiently represented with graphs. These methods are increasingly needed to reduce the enormous monetary and time costs of developing new pharmaceutical compounds [96, 98]. In particular, deep learning models for drug discovery [99] focus on computer aided drug design, where in silico experiments allow for a faster search and delivery of new candidate drugs [100]. Moreover, DNNs can be used to predict the properties of new compounds in silico, to estimate their activity levels in different settings, to predict their side–effects, and to generate candidate molecular structures [25]. This thesis focuses on drug side–effect predictors, and on a predictor of protein–protein interfaces, all developed with GNNs, that will be presented in Chapter 5, and Chapter 6 respectively.

2.4.3 Bioinformatics and GNNs

GNN applications to bioinformatics are not limited to drug discovery. As discussed in Section 2.4.1, graphs are commonly used in biology and medicine, and GNNs can be used to address a wide range of problems in these fields. Some examples of open problems that GNNs could potentially contribute to solve in the future are listed below.

- Protein folding prediction: the term protein folding refers to the pro-

cess by which proteins fold into their 3D structures, which determine their function. Predicting how proteins fold is an important problem in bioinformatics, as it can help to understand how proteins work and how they interact with other molecules. Traditionally, protein folding prediction has been approached using heuristics, which are methods that work well in practice but do not guarantee the optimal solution. More recently, DL methods, including GNNs, have been used to address this problem, as models can be trained on known protein structures to learn how proteins fold, and can then be used to predict the structures of novel proteins. Indeed AlphaFold is a recent AI system developed by DeepMind [101] that predicts a protein 3D structure from its amino acid sequence. It regularly achieves accuracy competitive with experiments. Nonetheless, the computational power needed to run AlphaFold is not available in common practice. Therefore, predicting protein folding is still a challenging problem due to the large number of possible conformations that a protein can take, which makes it difficult to find the optimal solution using traditional optimization techniques. However, AI methods, particularly GNNs, have the potential to provide a more efficient way to search the solution space and could lead to significant advancements in the discovery of new drugs and therapies.

- Protein–protein interaction prediction: predicting protein–protein interactions involves identifying which proteins bind to each other and determining the specific binding sites on the proteins [23]. This is an important problem in bioinformatics as protein–protein interactions are involved in many biological processes, including signaling pathways, gene regulation, and metabolism. There are several methods that can be used to predict protein–protein interactions, including clique detection or structural similarity analysis. GNNs can learn from known interactions to identify patterns indicative of protein–protein interactions and can be used to predict new interactions. Predicting protein–protein interactions can help to improve our understanding of biological processes and may also lead to the development of new drugs and therapies, for instance able to inhibit the formation of protein complexes which are fundamental for a virus transmission [102].

- AI–driven molecular dynamics simulations: molecular dynamics (MD) simulations are computer–based models that allow scientists to study

the movements and interactions of molecules over time. These simulations are an important tool in bioinformatics and other scientific fields, as they can help to understand and predict the behavior of molecules in different environments and under different conditions. Traditionally, MD simulations have been computationally intensive, requiring large amounts of memory and time to run. This has limited the length of time during which the dynamics can be simulated, making it difficult to study processes that occur over long time scales. GNNs have the potential to significantly improve the efficiency of MD simulations, since they can be used to reduce the memory and time requirements. This could lead to the development of new drugs [103, 24, 26] and therapies and improve our understanding of how molecules function in different environments.

- Multi–omics analysis: multi–omics data refers to data from multiple types of omics studies, such as genomics, transcriptomics, proteomics, and metabolomics. These studies generate large amounts of data that can be difficult to analyze and integrate. GNNs can be trained to identify patterns and relationships in the data and to predict properties of an organism or a tissue based on multi–omics information. This can be useful for understanding the mechanisms underlying biological processes and for predicting the behavior of molecules under different conditions.

GNNs are versatile and can be considered an attractive option to modeling many different biological problems [29]. Moreover, different models have been employed on different tasks, such as different types of convolutional or recurrent GNNs. Ad–hoc models can also be developed on a task specific basis or on a broader biological setting [83].

Chapter 3

Machine Learning Applications to Molecular Data

This chapter provides a thorough literature review concerning the research applications constituting the main focus of this thesis. Relevant literature on ML–based applications to drug discovery are introduced in Section 3.1, with a particular focus on drug side–effect prediction (Section 3.2). Eventually, Section 3.3 deals with the protein–protein interface prediction.

3.1 ML in Drug Discovery

In recent years, the drug discovery field has seen a significant shift towards the use of ML techniques [99], due to a number of factors, including the increasing complexity and cost of traditional drug development technologies, the growing availability of computational resources, and the rapid advancements in AI and DL. As a result, DL methods are being increasingly employed to enhance and even replace traditional processes in drug discovery [8]. One of the main areas where DL is being used in drug discovery is in silico experimentation, such as the identification of potential drug compounds [104], prediction of drug–target interactions [105], virtual screening [106], the analysis of binding pockets exploiting 3D CNNs [107] or druggability predictors based on ANNs [108], as well as reverse docking techniques to identify target proteins using a library of known drugs [109]. The use of DL in these areas has the potential to greatly improve the speed and efficiency of the drug discovery process, as well as increase the likelihood of success in

identifying effective drugs.

3.2 DSEs Prediction with DL

Drug side–effect prediction using ML is an important area of research in drug discovery. Predicting side effects of drugs in silico, before they are tested in humans, is a crucial task which can greatly improve the safety and efficacy of drugs, as well as reduce the risk of adverse events in patients. Many approaches has been proposed, from simpler methods based on Euclidean data [110, 111, 112, 113], to similarity scores between drugs [114]. The increasing in both quantity and heterogeneity of data, with the availability of processing resources, have helped in spreading ML–based approaches in this direction, from SVMs [115] and clustering techniques [116] to more complex predictors such as random forests [117] and deep MLPs [118] or NLP–based models [119]. Many methods that rely on biological information use protein-target as features, with the assumption that drugs with similar in vitro protein–binding profiles tend to exhibit similar side effects [120].

The prediction of DSEs requires knowledge of various biological entities, including genes, proteins, drugs, and pathways. The data used for such predictions is therefore naturally relational, thus being possible to represent it as a graph. GNNs have been found to be efficient in these types of scenarios; however, there is currently no known use of node–focused GNN–based approaches for predicting the side effects of individual drugs, although they have been employed in a similar but distinct context, specifically for the prediction of polypharmacy side effects — i.e. those resulting from the concurrent use of multiple drugs. Polypharmacy is nowadays a common practice in modern medicine which can lead to a large number of potential interactions, making it difficult to predict which combinations may lead to DSEs, given the complexity of the data. One approach to predicting polypharmacy DSEs using DL is to analyze a subset of possible drug pairs in the dataset, using graph attention networks (GATs) to measure the graph co–attention on the molecular graphs of the two drugs in each pair [121, 122, 123]. Another approach is to build a graph that accounts for the interaction between protein targets and drugs, and the known side effects of the single drugs. This graph is then analyzed using a GCN to predict the polypharmacy DSEs as links between drug nodes [7]. In this context, matrix factorization is a commonly employed technique for predicting links: in this cases networks are portrayed

as matrices with cells representing relationships between them. In such scenarios, link prediction can be treated as a problem of matrix completion [124] based on Singular Value Decomposition (SVD) [125] or Non–negative Matrix Factorization (NMF) techniques [126, 127]. Given the graph of drugs and DSEs, composed of two inner networks — a DSEs–drug bipartite sub–graph and drug–drug similarity sub–graph —, the task is then to predict the links in the bipartite network between drugs and DSEs by means of heat diffusion–based or similarity–based approaches [123]. In recent years, hybrid approaches have been developed to jointly learn drug features from both the macroscopic biological network and the microscopic drug molecules [128]: to predict potential connections between drugs and DSEs, the model calculates molecular structure embeddings and fingerprints based on the drug's SMILES, while side effects are represented as unique random vectors. In the corresponding bipartite graph, the GNN model is exploited to update nodes representation, which is eventually multiplied by the side effect feature matrix to recreate the adjacency matrix, solving the link prediction task.

3.3 Prediction of Protein–Protein Interfaces

Predicting protein–protein interfaces is crucial in structural biology and bioinformatics, since by identifying the specific residues that form the interface of a protein complex, researchers can gain a deeper understanding of the molecular interactions that drive biological processes. This knowledge can have significant implications for the development of new drugs, as it can help identify potential drug targets and design drugs that specifically bind to those targets. Furthermore, predicting the interface of a protein complex can also aid in the prediction of the overall 3D structure of the complex, which is crucial for understanding the function of the proteins involved. In addition, understanding protein–protein interactions can lead to the design of new proteins with improved properties: this can be used in the development of new enzymes for industrial applications and in the design of new vaccines.

Though detecting the interface of two monomers is important to predict the quaternary structure and functionality of proteins [129], this can be a hard task to be faced with traditional techniques [130]. Protein–protein interfaces can be predicted with a variety of approaches: based on sequence homology [131], Bayesian methods [132], analyzing combined docking sim-

ulations [133], or using SVM predictors [134]. Recently, GNN–based predictors have been developed for the prediction of molecular interactions [135, 136, 137], although no specific GNN methods have been reported for the detection of interfaces between monomers. In this context, the biological problem of detecting the interfaces between the two proteins can be reformulated as a maximum clique search problem [6], by constructing a so–called correspondence graph from the graph representations of the two interacting monomers, in which secondary structures elements are considered [5]. The interface will then correspond to the maximum clique in the correspondence graph [6]. Clique detection problems have already been addressed with GNNs [138], and, more recently, also in a transductive learning setup [139]. Finally, this strategy was also further refined by exploiting LGNNs [73].

Chapter 4

GNNkeras: A Keras–based Library for Graph Neural Networks

In this chapter, GNNkeras, the software framework for the implementation of GNNs, described in publication [**A2**], is described. This framework allows to easily define GNN models for homogeneous and heterogeneous graph processing, represented by GNN/LGNN and CGNN/CLGNNN (see Section 2.3.4) models, respectively. All the implemented GNN models can be used for classification, regression, and clustering on nodes, edges or whole graphs, both in inductive and mixed inductive–transductive learning settings [139]. In the original framework, GNNs are inductively trained based on a supervised learning environment. However, GNNs and LGNNs can also take advantage of transductive learning [140], thanks to the natural way the information flows and spreads across the graph. In the transductive framework, the training set nodes and their targets are used in conjunction with the test patterns. In particular, the feature vectors of a subset of the training nodes — called transductive nodes — are enriched with their targets, to be explicitly exploited in the diffusion process, yielding a direct transductive contribution.

The rest of this chapter is organized as follows: Section 4.1 introduces the motivation behind the development of GNNkeras, Section 4.2 describes the software in detail and its usability, then Section 4.3 draws conclusions on this work.

4.1 Motivation and Significance

In the context of ML research on graphs, it is important for researchers and software developers to have adequate and flexible tools that support the development of applications with current GNN models and possibly favor the study of new versions of GNNs. For this reason, a new Keras library was developed, based on the original GNN model [1], which allows to implement the whole subclass of recurrent GNNs [80], and LGNNs [73].
GNNkeras users can easily access a huge number of ML features. This fact is guaranteed by Keras, which is built on top of TensorFlow 2.x [141] and is one of the most used and complete software libraries for ML. As far as the author's knowledge goes, GNNkeras is the first tool specifically designed for implementing recurrent GNNs, while other tools exist for building models of the subclass of convolutional GNNs. Finally, GNNkeras is flexible in that it permits to manage a variety of activities, graph domains and learning approaches.

The characteristics of GNNkeras are many and can be summarized in the following points.

- GNNkeras allows to develop and deploy GNN models easily, in a few lines of code, and with high versatility. Representing a GNN as a GNNkeras model gives a considerable advantage compared to previous common solutions, which were manually written from scratch with TensorFlow.

- All the three different types of deep learning problems on graphs are implemented: node–based, edge–based, graph–based.

- GNNs can be layered, implementing the LGNN version for more complex problems.

- GNNs and LGNNs can be applied to heterogeneous graphs.

- All the three super–categories of deep learning tasks can be tackled with GNNs: regression, classification, and generation.

- Inductive and mixed inductive–transductive learning can be adopted.

Although there are already several excellent GNN libraries available, based either on PyTorch (PyTorch Geometric [142], Deep Graph Library

DGL [143], etc.) or Tensorflow (Spektral [144], etc.), GNNkeras has been developed as a new library, rather than contributing to the existing ones. GNNKeras was born to address specific use cases or applications which are not fully supported by existing libraries, in particular to handle heterogeneous graph data, which is not possible with the other Tensorflow–based libraries. For this reason, GNNkeras can provide a tailored solution, with specialized APIs and workflows, that are designed to make it more efficient and effective for users to build GNN models on heterogeneous graphs.

The expected impact of GNNkeras is mainly related to its capability of helping its users in speeding up the proposal of new research and the development of advanced software.

4.2 Software Description

The software implements the GNN model formulation described in Section 2.3.1, the CGNN model described in Section 2.3.4, and LGNNs [73] as described in Section 2.3.3.

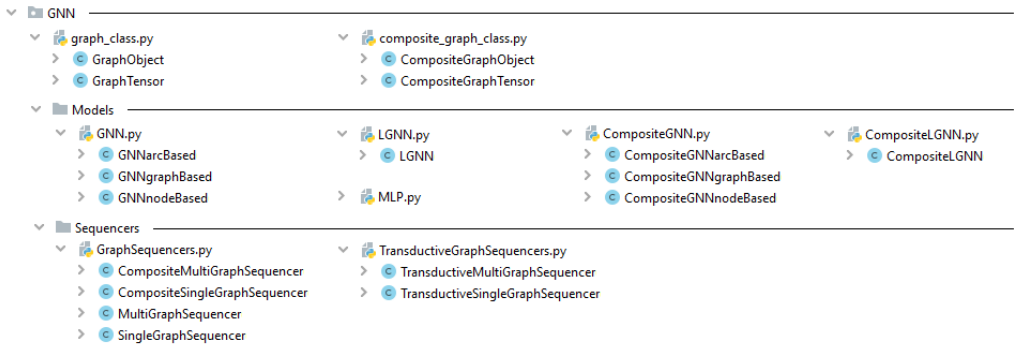


Figure 4.1: Software architecture: the main *GNN* directory contains graph data representation classes; the *Models* sub–directory provides MLP, GNN, LGNN, CGNN and CLGNN implementations; the *Sequencers* sub–directory provides graph sequencers for feeding models with GraphObject/CompositeGraphObject data. Note that the MLP model is a function which returns a Keras Sequential model, meaning that every Sequential model can be used for implementing the state transition network f_w and the output network g_w. In the latest version of the software, no distinction is made between multi–graph and single–graph sequencers, since only GraphSequencer/CompositeGraphSequencer and TransductiveGraphSequencer/CompositeTransductiveGraphSequencer are provided.

To parallelize software execution on modern CPUs and GPUs, all the operations are based on matrix multiplications. Fig. 4.2 shows the process-

ing scheme of a heterogeneous graph by a CGNN model implemented with GNNkeras. The homogeneous case is analogous to a CGNN with a single node type.

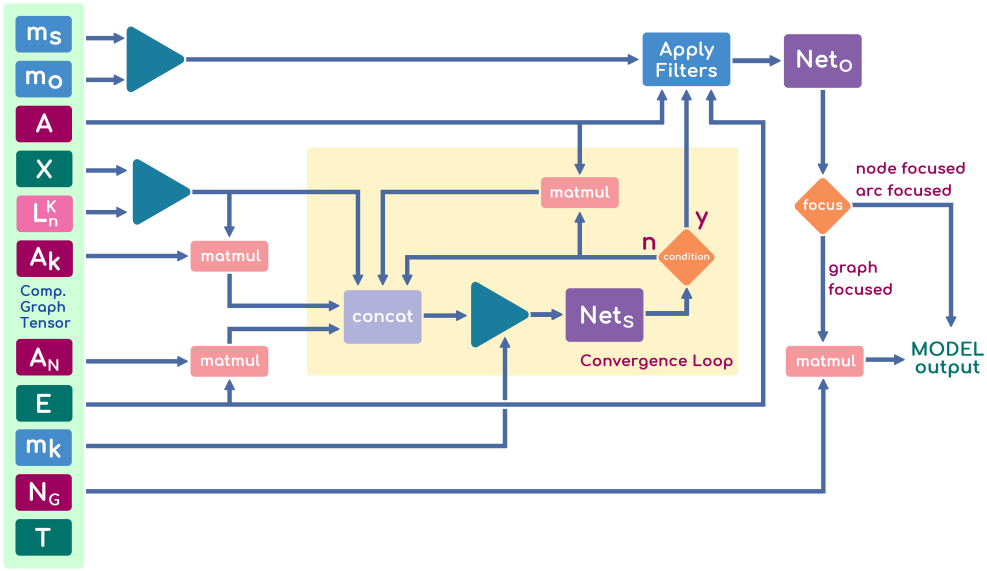


Figure 4.2: CGNN model software scheme. The GraphSequencer generates batches of GraphTensors which are presented to the model as input. All quantities pass through multiple operations (matrix multiplications, boolean mask filtering and concatenating processes) to form the input to f_w and g_w.

GNNkeras has been implemented as a module using the Python3 programming language, and it includes GNN models for node–focused, edge–focused, and graph–focused applications, working in homogeneous and heterogeneous graph domains, both in inductive and transductive learning contexts. It is based on NumPy, SciPy, and TensorFlow libraries, as NumPy and SciPy provide efficient numerical routines for dense and sparse data, while TensorFlow and Keras provide a simple and smart way to define and manage models, as well as to simplify the learning and evaluation processes. In particular, TensorFlow [141] is an open-source set of libraries for creating and working with neural networks, developed since 2017 by Google Brain, a deep learning and artificial intelligence research team from Google AI, the research division at Google formed in 2011 and dedicated to artificial intelligence. Since 2019 with its 2.0 version, TensorFlow has adopted Keras as its official high-level API, as it simplifies the implementation of complex neural networks with its easy to use framework. Developed by Google, Keras provides