UNIVERSITÀ
DEGLI STUDI
FIRENZE

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

Graph Neural Networks for Molecular Data

Candidate
Pietro Bongini

Supervisors
Prof. Monica Bianchini

Prof. Franco Scarselli

PhD Coordinator
Prof. Stefano Berretti

Cycle XXXIV, years 2018-2022

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

PhD Thesis Committee:

Prof. Alessandro Di Stefano

Prof. Markus Hagenbuchner

Prof. Marco Maggini

Prof. Stefano Cagnoni

Prof. Battista Biggio

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

Abstract

Graphs are a very important and common form of data representation, providing information on data entities and 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. Graph Neural Networks are Deep Neural Networks capable of processing graph structured data with minimal loss of information. They have recently known a steady growth, and have been applied to an increasing number of problems of different nature, leading to the development of new theories, models, and techniques. In particular, molecular data proved to be a very suitable application field for Graph Neural Networks, as many biological entities are naturally represented as nodes, edges, or graphs. In this thesis, three applications of Graph Neural Networks to molecular data, relevant both from the point of view of Deep Learning and from that of bioinformatics, are discussed. Molecular graph generation is an innovative task, in which Graph Neural Networks could help design the structures of new drug candidates for drug discovery research. Drug side–effect prediction is another challenging task: predictions are based on heterogeneous and complex data that can be processed with Composite Graph Neural Networks. Protein–protein interfaces can be detected by identifying the maximum clique in a correspondence graph of protein secondary structures, a problem that can be solved with Layered Graph Neural Networks. Moreover, a software framework for the model implementation was developed. These applications, inspired by real–world problems, constitute a very good testing ground for the development and evaluation of Graph Neural Network models. Very promising experimental results provide useful insights and allow to draw interesting conclusions on the capabilities of Graph Neural Networks in analyzing and generating molecular data.

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
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
RL : Reinforcement Learning
RNN : Recurrent Neural Network
SSE : Secondary Structure Element
SVM : Support Vector Machine
VAE : Variational AutoEncoder

Chapter 1

Introduction

Machine Learning (ML) is a research field in constant and steady growth, where solutions are found for problems of increasing variety and complexity. In the last decades, many breakthrough discoveries have opened new directions, most of which are far from being fully explored. In this thesis, the attention is focused on deep supervised learning on structured data, and, in particular, on the application of Graph Neural Networks (GNNs) [1] to biological tasks inspired by real–world problems.

1.1 Graph Neural Networks in Bioinformatics

Graph structured data are nowadays ubiquitous, covering topics ranging as far as social network analysis [2] and molecular property prediction [3], protein folding [4] and power network analysis [5]. The graph structure itself is an important part of the information, with graph nodes representing entities and graph edges accounting for relations between entities [6]. With traditional ML methods, though, it is not possible to process the graph structure in its native form. In order to feed a graph to a neural network, it is mandatory to obtain a Euclidean version of the data, to then feed the vectors to an unstructured model. Various methods to encode graphs, graph nodes, or graph edges, into vectors have been devised, but even the more conservative ones imply a relevant loss of information, concerning particularly the relational part of the data. GNNs, instead, can process the graph natively, with minimal loss of information [7]. Since they were first introduced [8], many

different model versions have been devised, and their successful applications to graph structured data have become countless [9].
Biological and chemical data are one of the most trending fields of application of graph–structured models [10]. Graphs describe very well many different aspects of chemistry, biology, and bioinformatics. For instance, the structural formulas of molecules are traditionally represented as graphs, in which each node represents an atom and each edge a chemical bond between two atoms. Larger molecules, such as polymers, can be described by graphs in which each node corresponds to a group of atoms. Protein structures are also represented as graphs, in which nodes correspond to aminoacids or secondary structures, depending on the scale of representation. Interaction graphs of genes, transcripts, or proteins, metabolic networks, pathways, and many other graph data can be cited along with these examples.
GNNs can be applied on any graph dataset, going from sets composed of many small graphs (e.g. drug structure databases) to a single graph representing a large network of entities (e.g. a gene interaction network). They can solve various types of problems: graph property prediction, edge property prediction, and node property prediction, each in both regression and classification settings. This makes these models extremely adaptable and, given also the aforementioned properties, 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 [11]; attention mechanisms can make the models explainable [12]; hierarchical versions can process large graphs with complex structures [13]; composite GNNs can process heterogeneous graphs [14].

1.2 Thesis Summary

This thesis is focused on GNN applications to molecular data, solving biological tasks inspired by real–world problems. A broader view on ML algorithms for structured data is provided. Biological problems on structured data are presented and discussed, in the scope of applying the previously mentioned ML algorithms, with a particular focus on GNNs. The relevant literature in this field is reviewed, in order to put the foundations for the presentation and discussion of the work.
The development of a software framework for the implementation of GNNs is discussed. The software is based on the well–known Keras ML library,

and it has been used in all the other research works presented in this thesis. It constitutes a key tool for the implementation of the original GNN model for a wide variety of possible scientific applications, including classification, regression, and generation tasks, that can be either node, edge, or graph focused. Three applications, which are relevant from the point of view of ML as well as from that of bioinformatics, are then illustrated.

The first is a GNN model for the generation of molecular structures: Molecular Generative Graph Neural Networks for drug discovery (MG^2N^2). This approach explores the possibility of generating molecular graphs sequentially and with GNNs. Generating molecular graphs of potential drugs can be a fundamental help for drug discovery processes, exploring the space of possible new compounds way farther from existing molecules than traditional techniques do. From the point of view of machine learning, graph generation is a challenging problem, and a less explored direction when compared to classification and regression tasks. Generating graphs is difficult because models have to deal with multiple decisions involving discrete entities (e.g. does a node exist or not?). Generative algorithms for molecular graphs are based on recurrent models and Reinforcement Learning (RL), while GNN–based generators are a promising but yet to be explored solution. Sequential generation consists of building the graph one step at a time, adding nodes and edges progressively, in contrast to single–step generation, in which a graph's adjacency and feature matrices are generated in one go. Sequential generation, though bringing issues related to node ordering and training biases, is generally more efficient for small compounds, and more explainable (the construction of single nodes can be analyzed). Sequential generative models for graphs are usually based on Recurrent Neural Networks (RNNs) or RL models, which respectively analyze the sequence of steps, or the sequence of decisions, needed to create the graph, instead of the graph itself. This work explores the possibility of developing a molecular graph generative model based on GNNs. The results demonstrate the advantages of using GNNs, that are capable of re–analyzing the graph at each step, exploiting all the available structural information for taking the next decision, obtaining relevant results in terms of quality and novelty of the compounds.

The second application consists in the prediction of Drug Side–Effects (DSE) 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 dur-

ing clinical trials. A dataset is built for this task, including relevant heterogeneous information coming from multiple well known and publicly available resources. The dataset consists of a single graph, composed of drug and gene nodes, and their interactions. Drug nodes are associated 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, we exploit transductive learning to better adapt the model to this setup. Given its nature, the problem is particularly interesting in the ML scope, providing an interesting application case of GNNs to a complex task (multiple non–mutual exclusive classes to be predicted in parallel), on heterogeneous data, and in a mixed inductive–transductive setting. The results show that encoding these data in a graph structure brings an advantage over unstructured data, and that GNNs exploit the given information better than concurrent models.
The third application consists in the prediction of protein–protein interfaces with GNNs. Simulating the structural conformation of a protein in silico is complex and resource demanding. A reliable method for predicting interfaces could improve the prediction of quaternary structures and of the functionality of a protein. Representing the interacting peptides as graphs, a correspondence graph describing their interaction can be built. The correspondence graph is then analyzed in search of cliques, that mark the position of interfaces. Clique detection is a challenging problem in this setting, as the data distribution is imbalanced (only few nodes belong to cliques) and because of 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.
These three applications, additionally to their relevance from the point of view of research on molecular data, are a good testing ground for GNNs. This thesis presents them in order of importance: the first application is the most innovative and challenging, since molecular graph generation is a complex problem and GNNs have not been employed in this task yet; the second application demonstrates the capabilities of GNNs on a heterogeneous relational dataset, built from multiple different data sources; the third application is more classical, yet it allows to face a relevant biological task, and to test GNNs in clique detection on an unbalanced dataset. Therefore

these three problems allow to obtain relevant results, and to draw interesting conclusions, from the point of view of ML.
Other works, mainly related to bioinformatics and structured data, are then briefly sketched and discussed. The rest of this Section summarizes the contributions of the thesis, presented in Subsection 1.2.1, and the thesis structure, described in Subsection 1.2.2.

1.2.1 Main Contributions of the Thesis

The main contributions of the thesis are summarized below:

1. A software framework, based on Keras, for the implementation of GNNs in multiple application scenarios, described in publication [P03].

2. A generative model for molecular graphs based on GNNs, described in publication [P01].

   (a) MG^2N^2, a modular, GNN–based framework for the sequential generation of molecular graphs.

   (b) Experimental results on two drug discovery benchmarks, placing the model among the state–of–the–art approaches.

   (c) An evaluation metric for the generation performances that aggregates pre–existing evaluation scores, accounting for both validity, novelty, and uniqueness of the generated compounds.

3. A GNN predictor of the occurrence of DSEs, described in publication [P02].

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

   (b) A predictor based on GNNs and developed on our dataset, that deals with a multi–class multi–label classification task on heterogeneous data, in a mixed inductive–transductive setting.

   (c) Experimental validation of the predictor, with interesting results highlighting the gap with a non–graph–based method.

   (d) Ablation studies on DSEs and data sources, that underline their importance for the learning process.

4. A GNN–based method for the identification of protein–protein interfaces, described in publication [P11]

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

   (b) A GNN predictor trained on our dataset, that 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 organized in chapters, sections, and subsections, in order to provide a clear structure for the explanation of the relevant aspects of Deep Learning (DL), methods for structured data, application examples of GNNs, and the other works carried out during the Ph.D.
After this introductive Chapter 1, Chapter 2 describes the general concepts of DL, its application to graphs and other data structures. A detailed description of the GNN model is given, along with its applications to biological problems, with a particular focus on drug discovery and bioinformatics.
Chapter 3 gives a review of the literature regarding the research fields relevant in the scope of this thesis, giving insights on theoretical and practical works concerning the application of GNNs to biological problems, and on the classical methodologies and state of the art in the tasks addressed in this thesis.
Chapter 4 presents one of the main contributions of the thesis: a software framework for the implementation of GNNs on a wide variety of possible problems. The framework is built on Keras, a well–known and efficient library for the implementation of DL models, belonging to the Tensorflow environment. Our framework provides a easy to use and lightweight tool for implementing the original GNN model in scientific applications, such as the other contributions described in this thesis.
Chapter 5 presents the most important contribution of the thesis: the development of a generative model for molecular graphs based on GNNs, and its application to two relevant drug discovery datasets. Insights on various aspects of the methodology, also concerning open problems in the field, are given. The experimental results are presented and compared to other state of the art methods, leading to interesting conclusions and future developments

of the presented approach.
Chapter 6 describes another main contribution of this thesis. It corresponds to the development of a framework based on GNNs for the prediction of DSEs. A relational dataset of drugs and human genes is built, in order to take into account all the heterogeneous data relevant for the prediction of DSEs. Well known publicly available reliable data sources are used in the dataset construction process. A GNN predictor is then trained and tested on this dataset, leading to very promising results.
Chapter 7 presents the last contribution of this thesis: a GNN model for the prediction of protein–protein interfaces. Given the graphs describing the two structures of a pair of peptides, a correspondence graph is built accounting for structural similarities and contacts. Clique detection on the correspondence graph is a reliable method for finding the interfaces between the two peptides. A GNN model capable of predicting cliques in this setting is presented, with interesting experimental results.
Chapter 8 briefly sketches the other works carried out during the Ph.D., not related to the main contributions of this thesis, yet still relevant to ML and bioinformatics. These include the development of an algorithm for the prediction of glycine–mutations of protein structures, in order to induce the formation of transient pockets that could interact with drug molecules. This approach is also exploited for the prediction of possible mutations that could disrupt the coronavirus spike protein structure.
Finally, Chapter 9 draws the conclusions of this thesis, suggesting the future developments of the works presented, and discussing their relevance and meaning.

Chapter 2

Deep Learning on Structured Data

This chapter introduces the basic concepts of DL, the applicability of DL techniques to structured data, and, in particular, to graphs. An insight on biological data for which DL approaches are particularly suitable is provided, introducing the three main applications described in this thesis. Section 2.1 describes DL in the wider framework of Artificial Intelligence (AI), gives an introduction to ML in general in Subsection 2.1.1, with a focus on Artificial Neural Networks (ANNs), and in particular Deep Neural Networks (DNNs), in Subsection 2.1.2, and the learning algorithms for training these models in Subsection 2.1.3. Section 2.2 introduces models and algorithms for structured data, describing the data types in Subsection 2.2.1, and the models in Subsection 2.2.2. Section 2.3 defines GNNs, sketching a general model in Subsection 2.3.1, how it learns in Subsection 2.3.2, a composite model for heterogeneous graphs in 2.3.3, a theoretical analysis of the approximation capabilities of GNNs in Subsection 2.3.4, and an overview of the principal models and their applications in Subsection 2.3.5. Finally, Section 2.4 describes how GNNs are applied to molecular data: Subsection 2.4.1 reviews biological problems involving graph data, Subsection 2.4.2 gives a deeper view on graph–based drug discovery tasks, and Subsection 2.4.3 introduces the applications of GNNs to these bioinformatics problems.

2.1 Deep Learning

DL is the branch of AI that studies how ML models with deep architectures learn to solve complex tasks. AI is the discipline that broadly studies computers that behave in an intelligent way, ranging from systems that simply replicate solutions designed by human experts, to ML models that learn their solutions from experience. 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 training can be either supervised, when the correct value $\hat{y}$ of $f(x)$ is known, or unsupervised, when the ML model learns from unlabeled data, using only the information attached to the examples themselves.

2.1.1 From Machine Learning to Deep Learning

The concept of ML was first introduced in a 1959 study [15], which can be considered one of the first works on learning algorithms. The initial works, though defining the basic concepts and ideas behind learning machines, lacked an efficient learning mechanism. As a consequence, the real breakthrough didn't come before the 1980s, with the introduction of the BackPropagation algorithm [16]. ML encompasses many different paradigms of learning machines:

- RL [17] studies how agents can learn to take actions for solving a problem, through a mechanism based on rewards (reinforcements);
- Support Vector Machines (SVMs) learn weights to discriminate data linearly, or with kernels that account for non–linearity [18];
- ANNs combine artificial neurons [19] to approximate non–linear functions of variable complexity.
- Self–Organizing Maps are a special type of ANN, based on competitive learning rather than inductive learning. They are an unsupervised model: neurons are organized into a regular grid (map) and learn to activate based on the input and on the activation of their neighbors [20].

- Clustering methods are unsupervised learning algorithms that can be trained to associate data entities by distance in the feature space [21].
- Random forests [22] learn to build ensembles of decision trees that fit training data according to supervisions.
- Gradient boosting techniques [23] realize a strong decision model by building an ensemble of several weak decision models (usually decision trees).

DL is based on early theoretical works that date back to the 1970s and 1980s. Before DL could know the fast and revolutionary development it has known in the 2010s (and is still ongoing at the present day), the issue to be solved were long–term dependencies, and the consequent vanishing gradient problem [24]. Many different solutions were proposed, with the development of models based on different paradigms, and designed for different data structures that will be overviewed in Subsection 2.2.2.

2.1.2 Deep Neural Networks

In the last decades, a particular class of supervised ML models, namely ANNs, became very popular for solving tasks of increasing complexity, outperforming other ML techniques more and more often. ANNs exploit the concept of artificial neuron: a simple processing unit that applies a linear or non–linear function to the weighted sum of its inputs [19]. Artificial neurons serve as building blocks of ANNs, which, depending on the number of neurons and their organization, can solve problems of variable complexity. The architecture of an ANN defines this organization, with the units being usually arranged into subsequent layers. A higher number of layers, and of units in each layer, increases the computational capabilities of ANNs, but also make training harder. In particular, the number of layers can be referred to as the depth of a network, and networks with more than one hidden layer (i.e. any layer which is not observable from outside, not being the input layer or the output layer) are called DNNs. The simpler type of ANN is the Multi–Layer Perceptron (MLP), which takes in input a vector of features and calculates an output function defined according to the problem under analysis. It was proved that MLPs are universal approximators on Euclidean data [25, 26, 27], meaning that any association between input and output vectors can be learned by an MLP with a sufficient quantity of units [26] and

at least one hidden layer. The four–layered Cognitron can be considered the first example of DNN [28]. More complex ANNs, and DNNs, were developed in order to process structured data, and will be examined in Section 2.2.

2.1.3 Learning with Deep Models

Feedforward models are the simplest type of ANNs, where the input signal is propagated through the N layers $L_1, L_2, ...L_N$, from input to output, obtaining the network output y. The output y is then compared to the supervision $\hat{y}$, by the means of an error function $E(y, \hat{y})$ (also called loss function, or cost function). E can be derived with respect to the outputs of the single neurons, obtaining the contribution to the error of each unit. The model is therefore optimized with an algorithm of the gradient descent family. The process starts from the calculation of the derivative $\delta E/\delta y_i$ for each unit i of the output layer L_N. These gradients can then be used to calculate the contribution to the error of the units belonging to the last hidden layer $\delta E/\delta y_j$ for each unit j of the hidden layer L_{N-1}, as shown in Eq. (2.2):

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

This process is repeated in cascade, down to the input layer, as a backward calculation of the contributions of each unit in the network, and is therefore called BackPropagation [16]. These contributions are then exploited to update the network parameters θ, in order to learn a better configuration according to the error. DNNs, where the number of layers N is large, are characterized by the long–term dependency problem [24]. Actually, it is extremely difficult to learn the dependencies between neurons located in distant layers due to the so called vanishing gradient problem: the derivative of the error gets smaller when backpropagating to lower network layers, up to the point in which the layers closer to the input cannot be trained from experience. This prevents traditional ML algorithms, like standard BackPropagation, from successfully training DNN models [24]. These issues were resolved by introducing ad–hoc backpropagation methods, activation functions like REctified Linear Unit (RELU), and batch normalization.

2.2 Machine Learning on Structured Data

Structured data are ubiquitous and have a role of increasing importance in our daily lives. As the world becomes more interconnected, the amount and heterogeneity of data regarding each single problem tends to increase, making relational data and data structures more important every day [9].

2.2.1 Structured Data Types

Euclidean data, in which each entity is described by a simple Euclidean vector of feature values, is the traditional data type processed by ML models and ANNs. Real–world data, though, can have a wide variety of different structures, yet they can be represented by some main categories:

- *Sequences* are the simpler type of data structure, representing each entity as part of a temporal or spatial succession of similar entities. Each entity can be followed by one and only other entity, and can follow one and only other entity.
- *Trees* are a generalization of sequences, in which each entity can be followed by more than one entity, yet each entity follows only one other (parent) entity.
- *Graphs* are a generalization of trees, in which entities are represented as nodes, and relationships of any type between entities are represented as edges.
- *Images* are regular grids, a particular type of graph, in which each pixel can be seen as a node, and edges are present only between nearby pixels.

These categories describe almost every type of data that can be found in the real world and online. Just to make some examples:

- Nucleic acids, proteins, temporal sequences of weather observations, item queues, are naturally represented as sequences.
- Phylogenetic trees, decision trees, hierarchies of entities are trees.
- Protein structures, metabolic networks, molecules, power grids, traffic systems, citation networks, knowledge graphs, social networks, and the internet are all examples of graphs.

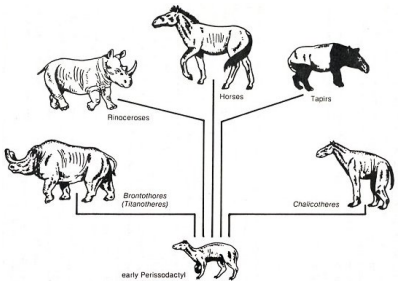
- Images can represent almost anything. Radiography and skin images help the identification of tumors, images of vehicles are exploited to analyze road usage, photos of human faces can be analyzed to detect emotions.

Visual examples of these structured data types are provided in Fig. 2.1. Often, combinations and hierarchies of these structures are found. Just to make some examples: 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, the weather can be analyzed using sequences of graphs.
Using traditional methods, entities belonging to structures, or belonging to structures of structures, are encoded into Euclidean feature vectors, exploiting ad–hoc algorithms. Even if these algorithms are usually optimized in order to retain as much information as possible, they all imply a certain loss. In Subsection 2.2.2, instead, we will analyze DL solutions for exploiting structured data in their natural form.

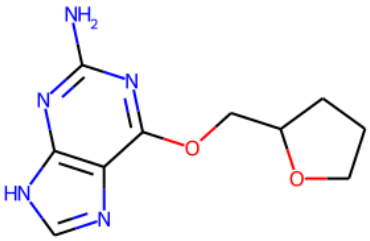
2.2.2 Structure–Oriented Models

DNNs can be defined with complex architectures, in order to adapt the model to the structure of the data it has to process. The first approaches in this direction date back to the early days of research on training neural networks with BackPropagation, as the BackPropagation Through Time algorithm (BPTT) [29] was published in 1990, just four years after the seminal work on BackPropagation for feedforward neural networks [16]. The first type of structured data to be processed with ANNs, as suggested by the concept of BPTT, were sequences, with the introduction of RNNs [30]. RNNs exploit the concept of recursion, replicating the same layer (or group of layers) over each element of the input sequence. The introduction of Long Short–Term Memories (LSTMs) in 1997 revolutionized this category of models, introducing the concept of cell gates: special units that can switch on and off signal (and therefore gradient) propagation, allowing the network to store information (memory) spanning an arbitrary number of time steps [31]. Gated Recurrent Units (GRU) are similar to LSTMs, but only have one gate per unit (the forget gate) [32]. RNNs can be used for tasks including: natural language processing [33], protein secondary structure prediction [34, 35], stock market prediction [36], motion recognition [37], and speech recognition [38].

(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.

The same application fields have been explored also with one–dimensional Convolutional Neural Networks (CNN-1D) [39], and with Transformers [40]. Transformers have revolutionized the field, by introducing an attention mechanism that can weigh the sequence elements by importance and overcome sequence ordering biases [40].

Images are traditionally more complex to process, as they can be a large input, with a size in the order of the number of pixels. Learning on images with an MLP is not efficient, as slicing the image to fit into a vector has a considerable cost in terms of structural information loss. Moreover, processing an image with dense layers is often impossible due to the size of the input (and consequently of the upper layers). To reduce the number of parameters and make a trainable ANN model that can process the pixel grid without slicing it, the breakthrough discovery was the concept of convolutional filter, employed for the first time in 2012 [41], and based on a theory from 1989

[42]. The convolutional filter takes in input a small patch of pixels, returning one single value, in an operation that can be repeated over the whole image. The operation can be carried out by a neuron, and combined with other convolutional filters in a convolutional layer. A Convolutional Neural Network (CNN) is formed by multiple convolutional layers, combined with pooling layers for the reduction of the input dimension, and possibly dense layers. A multitude of CNN architectures have been proposed, in order to apply them to different tasks, introducing deeper models as the research goes on [43]. The applications include but are not limited to: image classification [41], segmentation [44], object detection [45], image generation [46]. The depth of these models have reached the order of hundreds of layers, in particular after the introduction of networks with residual connections between layers. 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 [47]. CNNs are not limited to 2D images, and have been generalized to process 1D sequences, 3D images, and videos (creating recurrent CNNs).
1D, 2D, and 3D grids are particular types of graphs, yet generalizing CNNs to any kind of graph is impossible. Learning on graphs has always been a tough problem. Many methods have been devised to extract Euclidean information from graphs, like, for instance, techniques based on random walks [48]. A more advanced approach is that of learning kernels that can approximate functions on graphs [49], also with many applications to bioinformatics [50]. Some particular cases, like trees and directed acyclic graphs can be processed with models of the RNN family [51, 52]. The breakthrough discovery, in this field, was the theorization of neural networks that can process graphs by adapting their architecture to the input graph topology, namely GNNs [1]. These models will be extensively described and discussed in Section 2.3.

2.3 Graph Neural Networks

GNNs were first theorized in 2005, as networks that replicate the topology of the input graph, and exchange messages between neighbor nodes [8]. The seminal work, presenting the original GNN model in a full mathematical formulation dates back to 2009 [1]. GNNs typically act on a graph dataset D that can be composed of one or more graphs. In any case, even if graphs can be merged into batches for performance reasons, from the mathematical

point of view, each graph is processed independently, and we can therefore analyze the behaviour of the model on one generic graph $G \in D$ at a time. A graph $G = (V, E)$ is composed of the following:

- A set of nodes N,
- A set of edges $E \subseteq N \times N$,
- Labels, l_n $\forall n \in N$, describing the corresponding entities can be associated to the nodes,
- Labels, $e_{m,n}$ $\forall (m, n) \in E$, describing the corresponding relationships can be associated to the edges.
- Moreover, it is useful to define a neighborhood function Ne, that, based on the edges E, associates each node n to the set of its neighbor nodes $Ne(n) \subset N$.

2.3.1 The Graph Neural Network Model

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 is associated to each node $n \in N$, and iteratively updated by a learnable state updating function f_w. The state is a vector of dimension d_x, set as a hyperparameter of the GNN, and initialized by sampling from a random distribution usually centered on the origin of $\mathbb{R}^{d_x}$. The number of state update iterations K is determined as a hyperparameter of the model, as well. Given the randomly sampled initial states x_n^0, $\forall n \in N$, the state of any node n at iteration t can be calculated with f_w as in Eq. (2.3):

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

In Eq. (2.3), M is the function defining the message sent from each neighbor $m \in Ne(n)$ to node n, while A is the aggregation function of all the messages coming from the neighbors $Ne(n)$. In principle, M can be any function that returns a vector (the message) 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 the works concerning 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 edges are not labeled) from the computation:

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

In principle, 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 d_m. 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 it is the case in the (convolutional) aggregations of GraphSAGE [53]. In the works discussed 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, the hyperparameter a can be defined to select the aggregation function, that takes values $1/|Ne(v_i)|$ or 1, for obtaining the average or the sum, respectively. Combining all these concepts, we can rewrite the state updating function in its general final form, as in Eq. (2.6):

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

Once the maximum number of iterations K is reached, the final versions of the node states $x_n^K,\ \forall n \in N$, are fed in input to the output network, which approximates the output function g_w. In the following, we will define g_w for the three types of problems addressed by GNNs (node based, edge based, graph based). Once again, we define the general forms of g_w, with the possibility of excluding the labels from the calculation in some application scenarios.
In node based 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):

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

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

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

Finally, in graph based 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):

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

2.3.2 Learning with Graph Neural Networks

The state updating function f_w in Eq. (2.6) is learned and implemented with an MLP, namely the state network. Another MLP, namely the output network, learns to approximate g_w. These two MLPs are used as building blocks, and replicated over the topology of the graph in order to obtain the so called 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.

The output network takes in input the node state after the last iteration of state updating, and is therefore linked in cascade to the state network. Moreover, copies of the state network are replicated over each state updating iteration, and weight sharing is exploited also in this time dimension. As a consequence, our encoding network can be unfolded in time, obtaining a network with the same topology of the input graph, with K copies of the state MLP on each node and a copy of the output MLP on each node (if the problem is graph focused or node focused) or on each edge (if the problem is edge focused). A sketch of the unfolded encoding network is provided in Fig. 2.2.

Please note that the unfolded encoding network corresponds to a DNN with recurrent layers. In particular, given the numbers of layers in the state MLP L_f and in the output MLP 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

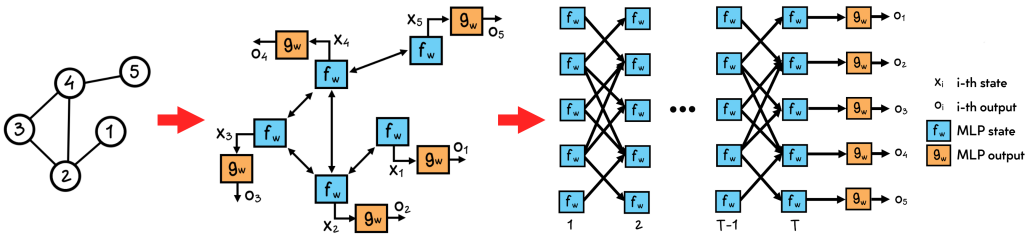


Figure 2.2: Sketch of how the GNN model produces the encoding network and its unfolded version on an example input graph (for a node focused problem).

the network, and therefore implement the learning process, it is sufficient to apply a regular optimization algorithm based on BackPropagation. Typical examples include stochastic gradient descent or the Adam optimizer [54]. 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 of the parameters of the network, averaging the contribution over all the replicas of each parameter, and applying the resulting weight differences.

2.3.3 Composite Graph Neural Networks

Composite Graph Neural Networks (CGNNs) are a particular type of GNN that can process heterogeneous graphs [14]. Heterogeneous graphs are often used to represent information about different types of entities interacting in multiple modes. Typical examples of this setting are knowledge graphs, in which entities of different types, and often with different features, need to be encoded into a single relational graph. Molecular graphs can also be seen as heterogeneous graphs, by distinguishing atom species as different node types.

CGNNs label edges representing different types of relationships with the one–hot encoding of the relationship type. If edge features are present, these are concatenated to the edge labels. Node types instead are treated as subsets of the node set N, and each type has a dedicated state network. As a consequence, given the number of node types nt in the dataset, the model employs nt different state networks to build its encoding network. Each state network F_i learns its own version $f_{w,i}$ of the state updating function in Eq. (2.6). The output dimension of each F_i is the same and corresponds to the state dimension d_x. To allow nodes to communicate through messages

which are coherent between different types, the node label is not 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, 2.9), are still valid instead, as each node has the same state dimension. The only modification consists in not sending the node label l_n in input to the network implementing g_w. Please note that this allows, in all the three cases (node, edge, or graph focused problems), to use a unique output network even in the most general case, in which N_{out} contains nodes of all the types defined in the dataset.
The learning process described in Subsection 2.3.2 still holds. The only difference, in the heterogeneous setting, consists in the fact that the encoding network, and consequently the unfolding network, are composed of nt state networks and one output network as building blocks, as represented in Fig. 2.3. As a consequence, the parameters the GNN has to learn are distributed in $nt + 1$ MLPs, in contrast to the two MLPs of the homogeneous setting.

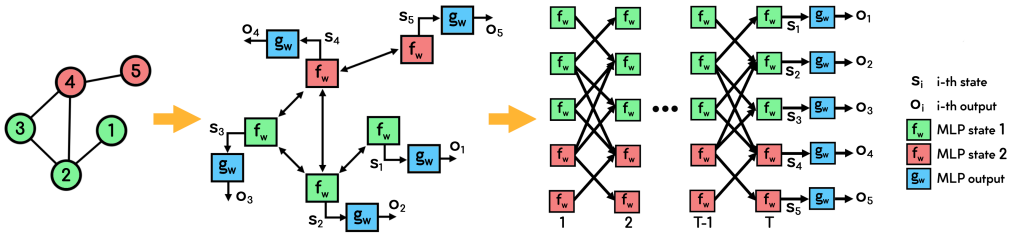


Figure 2.3: Sketch of how the CGNN model produces the encoding network and its unfolded version on an example input graph (for a node focused problem). Different node types are represented by different versions of the state network (green and red).

2.3.4 Approximation Power of Graph Neural Networks

The approximation capabilities of DNNs, and in particular MLPs, have been studied and described in various theoretical works [26, 25, 55]. GNNs,

though, operating on the graph domain, have to withstand different challenges, also concerning symmetries in the data structures. The computational power of GNNs was analyzed in a work [7] published in parallel with the original model [1]. Given a graph $G = (V, E)$, 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. The unfolding tree obtained is described in Eq. (2.11):

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

The unfolding tree T_n^k represents all the information on node n available to the GNN after k iterations. Therefore, 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, for k larger or equal to the diameter of G. 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) [7].
An alternative method to evaluate the computational power of GNN models was recently introduced in [56]. It is based on the Weisfeiler–Lehman graph isomorphism test [57], which associates a canonical form (based on node adjacency) to each graph, and recognizes two graphs as isomorphic if they share the canonical form. The one dimensional Weisfeiler–Lehman test (1–WL) cannot distinguish all the possible pairs of graphs, because the same canonical form can correspond to multiple non–isomorphic graphs. Therefore higher order tests are defined: the D–dimensional test (D–WL) is based on tuples of nodes of dimension D. The capacity of distinguishing non–isomorphic graphs grows with D.
GNN models can be classified according to their capability of simulating the Weisfeiler–Lehman test. Models that can simulate the 1–WL test are classified as 1–WL (at least as powerful as the one–dimensional Weisfeiler–Lehman test). Many currently used models are less powerful than 1–WL, as they fail to simulate the test. Interestingly, the 1–WL test is analogous to an iteration of neighborhood aggregation in recurrent GNNs: consequently, these models have been demonstrated to be all of class 1–WL, provided they

use injective neighborhood aggregation and output functions [56]. Currently, GNNs cannot simulate higher order Wesifeiler–Lehman tests, and no model has been classified as 2–WL or greater [56], but some efforts have been made in the direction of higher order GNNs, which require non–local neighborhood aggregation [58]. Moreover, unfolding trees and the Weisfeiler–Lehman test have been demonstrated to be equivalent methods for the evaluation of the approximation power of GNNs [59].

2.3.5 Models and Applications of Graph–based Models

Since the original GNN model was introduced, in 2009, many models of this class have been introduced [60]. Just the following year, the Layered Graph Neural Network (LGNN) model was introduced: LGNNs are composed of multiple GNNs connected in cascade, in order to progressively refine the model's output and obtain a greater learning capability with respect to a single GNN [61]. Models of the GNN family can be classified in two broad subfamilies: recurrent GNNs, and convolutional GNNs. The former, which also include the original model [1], are based on message passing between nodes and the recurrent calculation of node states. The latter, 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.
Examples of recurrent GNNs are Graph Nets [62], Gated Graph Sequence Neural Networks [63], Message Passing Neural Networks [64], and Graph Isomorphism Networks [56]. The first convolutional GNNs to be introduced were standard GCNs [65], followed by spectral convolution models [66, 67]. GraphSage generalized the concept of convolutional GNN by introducing different types of neighbor aggregation [53]. GCNs were also combined with attention mechanisms in Graph Attention Networks [68] and subsequent works, improving the predictions with information on important relationships [69] and dealing with explainability issues [12].
Graph–based models can be applied on any type of graph, in real world or synthetic problems. In the web domain, GNNs have carried out tasks of spam detection [70], community detection [65], sentiment analysis [2], content interaction prediction [71]. GNNs can be employed to predict logical

relations in knowledge graphs and future links in citation networks [72], and as recommendation systems [73, 74]. They have solved many node classification and regression tasks, as well as graph classification and regression tasks [9], also in a heterogeneous setting [75]. Models of the GNN family have showed performances at or close to the state of the art in problems of graph matching [76], weather forecasting [77], power grid analysis [5], and many others.
In the biological domain, GNNs can calculate molecule properties [64, 78], predict protein–protein interfaces [4], and classify compounds according to their mutagenicity [3] or activity against the HIV virus [79, 80].

2.4 Biological Problems on Graphs

The introduction of computational methods has opened new discovery routes in biology and medicine, leading to the development of interdisciplinary fields of research like bioinformatics and medical informatics. DL techniques are increasingly popular in these fields, providing many interesting solutions to previously untreatable problems, cutting the costs and times of traditional methods, and rethinking existing processes in a more efficient way.

2.4.1 Graph Data in Biology

In the biological domain, graphs are a very useful and diffused form of data representation. Just to start, structural formulas of molecules have always been perceived as labeled graphs, in which each atom corresponds to a node, whose label accounts for the atom type, and each chemical bond to an edge, whose label is the type of bond. These structures are also known in the literature as molecular graphs. The molecular graphs allow to predict many properties of each molecule, such as mutagenicity, anti–HIV or anti–cancer action, and other levels of activity.
More complex structures, such as polymers, proteins, and nucleic acids, are also best represented as graphs. Nodes can correspond to different structural levels, such as substructures, protein secondary structures, or DNA blocks. A hierarchical model can also be exploited, in which each node can be expanded into its substructure graph. Edges correspond to the interactions between these components. These data are often fundamental for solving important biological problems, such as predicting protein–ligand, protein–protein, and

protein–nucleic acid interactions.
Graphs are also exploited to represent logical information (i.e. knowledge graphs of biological entities). In this case, each node corresponds to an entity, and each edge to a relationship between two entities, while models can be trained to predict new biologically relevant relationships between entities (typically in a heterogeneous setting) given the known ones.

2.4.2 Graphs in Drug Discovery

Drug discovery is the discipline that studies how to develop new drug compounds. Discovering a new molecule is a long and expensive process [81], often involving researchers, companies, and national agencies [82]. Moreover, new discoveries become rarer and more expensive every year [83]. As a consequence, computational methods are required to innovate the process, to reduce the costs of development, and even to allow the discovery of new molecules that could not be devised with traditional methods [84]. In this scope, AI techniques, and in particular DL methods, are increasingly employed to find new solutions for a variety of relevant problems. 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 [85]. Since drugs and other molecules are efficiently represented with graphs, these tasks are all potential applications for GNNs. In this thesis we focus on a molecular graph generator, on a drug side–effect predictor, and on a predictor of protein–protein interfaces, all developed with GNNs, that will be presented in Chapter 5, Chapter 6, and Chapter 7 respectively.

2.4.3 Bioinformatics and Graph Neural Networks

GNNs applications to bioinformatics problems go beyond the field of drug discovery. As discussed in Subsection 2.4.1, graphs are ubiquitous in biology and medicine, the potential applications of GNNs being therefore uncountable. Some examples of relevant open problems, in which GNNs can substantially contribute in the future, are listed in the following.

- Protein structure folding: GNNs can be employed to predict how protein 3D structures fold. This problem has a finite yet enormous solution space, which has been explored mainly with euristics so far. The

introduction of AI–based methods has already brought unexpected improvements [86].

- Protein–protein interaction prediction: this problem can be tackled with different methods, like clique–detection on the interface graph, or by analyzing the structural similarity of the two proteins.

- AI–driven molecular dynamics: molecular dynamics simulations are incredibly complex, with high costs in terms of memory and time. As a consequence, they usually span times in the order of fractions of microseconds, but longer simulations would help understand and replicate in silico many cellular processes. Exploiting a neural network, and in particular a GNN, would reduce the memory and time costs of the simulations, also allowing longer time spans.

- Multi–omics analyses: in this case, multiple graphs need to be processed, one for each omics, in order to predict different properties of an organism, or a tissue.

GNNs are versatile and can be considered an attractive option to modeling many different biological problems [14]. 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 [64].

Chapter 3

Machine Learning Applications to Molecular Data

This chapter provides a thorough literature review concerning the research applications that constitute the main focus of this thesis. Section 3.1 introduces the relevant literature concerning the application of ML techniques to drug discovery. Section 3.2 focuses on graph generation, and the works related to the contribution presented in Chapter 5. Section 3.3 describes drug side–effect prediction and the works related to the contribution described in Chapter 6. Section 3.4 deals with protein–protein interface prediction and the works related to the contribution discussed in Chapter 7.

3.1 Machine Learning in Drug Discovery

In the last decades, the increasing complexity and cost of drug development technologies, the rapidly growing availability of computational resources, and the enormous progress of AI techniques on the spur of the development of DL algorithms, have brought a novel approach into the field of drug discovery [87]. This corresponds to an increasing employment of DL methods to boost, and in some cases replace, traditional processes [10]. Obviously, this revolution mainly concerns the in silico experimentation methods, such as the identification of candidate drug compounds [88], the prediction of drug–target interactions [89], virtual screening [90], the analysis of binding pockets exploiting 3D CNNs [91] or druggability predictors based on ANNs [92], and even reverse docking for the identification of target proteins using a library

of known drugs [93].

3.2 Molecular Graph Generation

One of the most important novelties brought by DL techniques is the possibility of generating structural formulas of new drug compounds from scratch [94]. This technique can be tailored to the objectives of a specific study by optimizing the generated compounds according to the desired properties [95]. This new way of building potential candidate compounds is substantially different from the traditional techniques, which usually started from a similar known molecule rather than generating structural formulas from scratch [10]. The generation of molecular graphs is a complex task, which can lead to the development of new instruments for drug discovery, potentially cutting the huge costs, in terms of both time and money, of this fundamental research process [81].
Graph generation is a complex problem also from the ML point of view, with several real–world applications, and many different approaches devised for solving it. Classical methods resorted to random mathematical models: The Erdös–Rényi model [96] was the first approach in this direction. Since Erdös–Rényi graphs tend to have unrealistically low clustering coefficients, especially with respect to community graphs, two methods were later developed, mainly to obtain more realistic small–world networks: the growth–based Barabási–Albert model [97], and the Watts–Strogatz rewiring model [98]. The recent developments in ML have stimulated its application to the field of graph generation. DL techniques, indeed, can capture the characteristics of a given domain from a set of examples, which are then exploited to generate new graphs. Variational Auto–Encoders (VAEs) [99] were the first neural network models to be employed for this purpose [100, 101]. The success of Generative Adversarial Networks (GANs) [46] in image generation has led to replicate the same adversarial approach for graph–structured data [102, 103]. This approach can be improved by adding constraints to the adversarial learning [104]. The different nature of the problem, though, has encouraged the development of alternative solutions as well. While VAEs, by sampling representations from a continuous latent space, can generate graphs as unitary entities, many methods tackle the problem with a sequential approach. The construction of a graph becomes, therefore, a sequence of decisions, in which a node or a group of nodes is added to the graph at

each step. On the one hand, many methods make use of RNNs to handle the decision sequence [105, 106, 107]. On the other hand, GNNs [1], with their capability of processing graph–structured data without loss of connectivity information, allow to build very powerful generative models. In particular, at each step, GNNs can exploit all the information contained in the partial graph generated by the previous steps, while recurrent models typically rely only on the sequence of previous decisions. In principle, this holds true for any GNN model, but the GraphNets–based DeepGMG is the only explorative approach in this direction [108], so far.
The space of molecular graphs is virtually infinite, and even constraining the size of molecules to few atoms, the number of theoretically possible compounds is overwhelming. Efficient automatic generative techniques are required for the exploration of such huge space, and deep generative models represent ideal candidates. Using SMILES notation [109], molecules can be generated as sequential objects. This approach has been carried out with VAE models [94], also exploiting the grammar of the SMILES language [110] [111], or even using SMILES fragments as words [112]. However, SMILES strings do not preserve the full connectivity information, as molecules are more naturally represented as undirected graphs, with finite (and relatively small) sets of vertex and edge types. Graph–based VAEs have been employed in the generation of molecular graphs [113]. Junction–Tree VAEs build graphs by connecting pre–extracted structural motifs [95], an approach which has been extended to larger molecules and polymers by making the VAE hierarchical [114]. This approach can also be improved by exploiting the valence histogram of each atom [115]. Recently, statistical flow models, characterized by an invertible encoder/decoder, have been developed [116]. Graph GANs have been employed for the generation of molecules, handling the decisions with RL techniques [117] [118]. Also the above mentioned DeepGMG has been applied to this task with promising results [108].

3.3 Drug Side–Effect Prediction with Deep Learning

Another interesting novelty brought by DL techniques to drug discovery is represented by the prediction of the occurrence of DSEs in silico. From simpler methods based on Euclidean data [119, 120], and similarity scores

between drugs [121], the approaches in this direction have increased the quantity and heterogeneity of data, in search of information on which to formulate more accurate predictions. This also implied applying ML techniques. The first approaches were based on SVMs [122] and clustering [123]. Methods based on chemical fragments embed drug structural information into Euclidean vectors [124]. These information were integrated and analyzed with predictors based on random forests [125] and deep MLPs [126]. Predicting the occurrence of DSEs involves knowledge on various types of biological entities, such as genes, proteins, drugs, and pathways. This means that data on which the predictions are based is inherently relational, and graph–structured. GNNs perform very efficiently in this scenario, but we are not aware of GNN–based approaches for the prediction of side–effects of single drugs so far. They have been used, though, in a related yet different setting: the prediction of polypharmacy side–effects. This task consists in predicting the side–effects triggered by the combined administration of two drugs. Two main GNN–based methods have been published in this scope: one analyzes a subset of the possible pairs of drugs in the dataset, applying GATs [68] to measure the graph co–attention on the molecular graphs of the two drugs of each pair [127]; the other builds a graph accounting for the interaction between protein targets and drugs, and the known side–effects of the single drugs, which is then analyzed with a GCN [65] that predicts the polypharmacy side–effects as links between drug nodes [128].

3.4 Prediction of Protein–Protein Interfaces

Detecting the interface of two monomers is fundamental to predict the quaternary structure and functionality of proteins [129]. These two characteristics are fundamental for studying the protein targets of drugs, and are difficult to simulate with traditional techniques. As a consequence, a reliable and fast method to predict them would significantly enhance current drug discovery techniques [130]. Protein–protein interfaces can be predicted with a variety of approaches: based on sequence homology [131], Bayesian methods [132], analyzing combined docking simulations [133], or using SVM predictors [134]. Generalizing to molecular interactions, predictors have been very recently developed based on GNNs [135, 136], but we are not aware of more specific GNN methods for the detection of interfaces between monomers.
Protein–protein interface detection can be reformulated as a maximum clique

problem [137], by constructing a correspondence graph based on the graphs of secondary structures of the two peptides [138]. The interface will then correspond to the maximum clique in the correspondence graph [137]. Clique detection problems have already been addressed with GNNs [139], and, more recently, also in a transductive learning setup [140]. Finally, this strategy was also further refined by exploiting LGNNs [61].

Chapter 4

GNNkeras: A Software Framework for Graph Neural Networks

In this Chapter, GNNkeras, a software framework for the implementation of GNNs, presented in publication [P03], is described. GNNkeras is a flexible tool: the implemented GNN models can be used for classification, regression, and clustering on nodes, edges or whole graphs. Additionally, GNN–based graph generative models can be built with this framework. Moreover, GNNkeras allows to build not only standard GNNs for processing homogeneous graphs, but also CGNNs for processing heterogeneous graphs (see Subsection 2.3.3), exploiting both inductive and mixed inductive–transductive learning [140]. Finally, LGNNs [61] can be instantiated, in both the homogeneous and the heterogeneous settings. 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 and its usability, and 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 [60], and LGNNs [61].
GNNkeras users can easily access a huge number of ML features. This fact is guaranteed by Keras itself, 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 we know, 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.

4.2 Software Description

The GNNkeras software is based on TensorFlow 2.x and Keras (TensorFlow backend), one of the most used DL frameworks worldwide [141]. The software implements the GNN model formulation described in Subsection 2.3.1, and the CGNN model described in Subsection 2.3.3. Moreover, LGNNs [61] can be implemented as well, by stacking GNNs one on top of the other, each refining the output of the previous GNN layer.
Furthermore, the mixed inductive–transductive learning scheme [140] applied in publication [P02] (see Chapter 6) can be used. In some applications, GNNs and LGNNs can take advantage of transductive learning [142], thanks to the natural way the information flows and spreads across the graph. In the mixed inductive–transductive framework, the training set nodes and their targets are used in conjunction with the test patterns. In particular, the labels 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.
GNNkeras has been implemented as a module using the Python programming language. It is based on NumPy, SciPy, and TensorFlow libraries. 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. Fig. 4.1 shows a graphical representation of the package directory organization.
The software relies on a custom graph representation, named GraphObject. For speeding up the learning procedure, a GraphObject is converted in another custom graph representation, called GraphTensor, which contains a

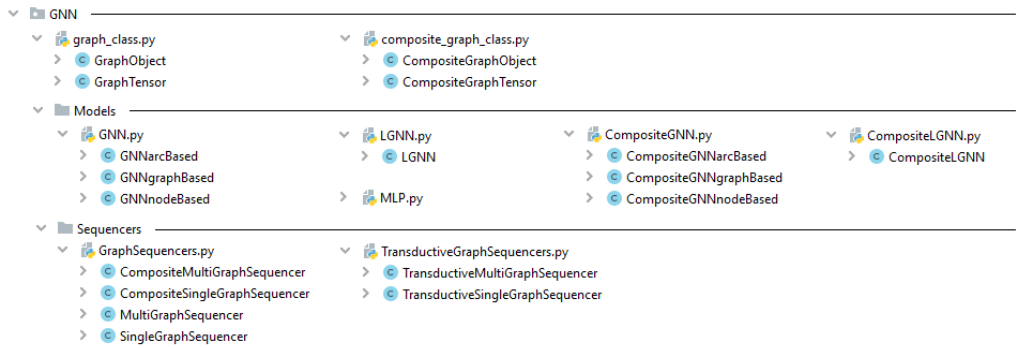


Figure 4.1: Software architecture: the main *GNN* directory contains graph data representation classes; the *Models* sub–directory provides MLP, GNN, LGNN, CGNN and CLGNN (Composite Layered GNN) implementations; the *Sequencers* sub–directory provides graph sequencers for feeding models with GraphObject/CompositeGraphObject 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.

tensor–based description of all the attributes for the graph to be correctly processed by the GNN model. In the heterogeneous setting, another class defined by CompositeGraphObject and CompositeGraphTensor is provided. A GraphObject or a CompositeGraphObject can be saved in a single NumPy npz file, or as a subdirectory of text files, which includes all the necessary matrices for their complete representation. Given a dataset of graphs, in the form of a list of graph data elements, these classes also provide a smart way to save the entire dataset in a single directory, from which it can be loaded when needed. In order to be correctly processed by the GNN models, GraphObjects and CompositeGraphObjects are required to be fed to a special data handler, the GraphSequencer. A total of six versions of GraphSequencer are provided: for multi–graph and single–graph–based datasets, in the homogeneous and heterogeneous graph domains, and for inductive and transductive learning approaches. It is worth noting that the transductive one is a special class of GraphSequencers, which is fed with homogeneous GraphObjects while generating heterogeneous graph data. For each epoch and batch, it splits the graph training nodes into two subsets of inductive and transductive nodes, thus generating two types of nodes being represented by a CompositeGraphTensor for the CGNN learning process.

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.

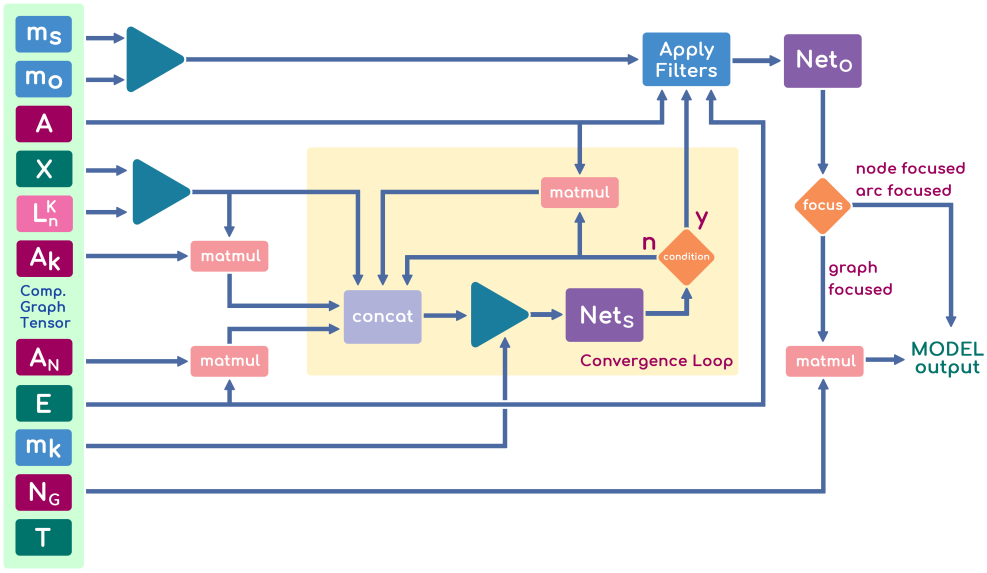


Figure 4.2: CGNN model software scheme. The GraphSequencer generates batches of GraphTensor which are presented to the model as input. All quantities pass through multiple operations (matrix multiplications, boolean mask filtering and concatenating processes) to form the input to f_w and g_w.

4.3 Conclusions

In this Chapter, GNNkeras, a new general GNN programming framework has been presented, which provides multiple Keras–based GNN models for homogeneous and heterogeneous graph processing, for both inductive and transductive learning approaches. GNNkeras has been designed with the aim of simplifying the programming procedures in the scope of research on recurrent GNNs. 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. We think that, due to the mentioned characteristics, GNNkeras is a flexible and suitable tool to exploit ML for graph data. The library can be used by researchers in ML to test new models and to design new applications. It can also be used by software developers

from companies and organizations designing applications for relational data. Finally, the exceptional interest in ML for graphs is a measure of the size and growth of the community operating in the sector and for which GNNkeras can be useful.

Chapter 5

Molecular Generative Graph Neural Networks for Drug Discovery

In this Chapter, discussing the contribution described in publication [P01], we present a sequential molecular graph generator based on GNNs [1], which we call Molecular Generative Graph Neural Network (MG^2N^2). A single node is added and connected to the graph, at each step. The method focuses on one node at a time, and generates its neighbors before processing the following node, preventing disconnected components from being created. Similarly to GraphRNN [105], we follow a Breadth First Search (BFS) ordering to decide which nodes to expand first. Edges are generated in parallel rather than sequentially, making the approach less computationally demanding. The control flow in MG^2N^2 depends on decisions implemented by three GNN modules. The sequential and modular nature of our method makes it interpretable. As previously mentioned, at each step, GNNs exploit all the information contained in the subgraph generated until that step. Gumbel softmax [143] output layers allow the networks to be trained over discrete stochastic distributions. Moreover, the modules are trained independently of each other: This feature simplifies the learning process and allows to retrain each module independently. The GNN model used in this work was derived from the original GNN approach [1], which was proved to be a universal approximator on graphs [7], a property which ensures that the GNN model is general enough to be able to make the complex decisions that the modules must implement.

The contributions presented in this Chapter is a new sequential generative model for molecular graphs, MG^2N^2, a new metric for molecular graph generation that combines into a single measure the scores accounting for validity, uniqueness, and novelty of the generated compounds, and the experimental evaluation of MG^2N^2 on two well–known benchmarks for the generation of small organic molecules, the Quantum Machine 9 (QM9) and Zinc datasets. The main novelty of the presented approach consists of using the GNN framework for molecular graph generation, with a modular architecture, in order to maximize the information and make the generative model flexible. The results show that the proposed approach outperforms very competitive baselines in the task of unconditional generation. The experiments also clarify the main properties of the method and show that MG^2N^2 is capable of generating molecules with chemical characteristics similar to those of the original datasets.
The rest of this Chapter is organized as follows. Details on how the GNN model described in Subsection 2.3.1 is implemented in MG^2N^2 are given in Section 5.1. Section 5.2 presents and discusses the generative algorithm in Subsection 5.2.1, its implementation with GNNs in Subsection 5.2.2, the graph preprocessing steps in Subsection 5.2.3, and issues about node ordering in Subsection 5.2.4. The experiments and their results are described and commented in Section 5.3: the datasets are described in Subsection 5.3.1, the experimental setup in Subsection 5.3.2, Subsection 5.3.3 provides details of the evaluation procedure, and, finally, Subsection 5.3.4 discusses the relevance and meaning of the results. Conclusions are drawn in Section 5.4.

5.1 Model Implementation

In this Section, we provide implementation details, for the study presented in this Chapter, of the model described in Subsection 2.3.1.
In particular, given the problem under analysis, the GNNs process molecular graphs, in which nodes correspond to atoms, and edges correspond to chemical bonds. Each node is labeled with the one–hot encoding of the atom type, and each edge is labeled with the one–hot encoding of the bond type. Therefore, the state updating function f_w takes in input both node and edge labels, and is implemented exactly as in Eq. (2.6).
Both aggregation functions (sum and average) described in Eq. (2.5) are used in the experimentation. In the experimentation described in this Chapter,

the problem of graph generation is divided into three classification subtasks. One of them is node–based while the other two are edge–based. The output function for the node classification task is the one formulated in Eq. (2.7), while the two edge classification tasks use the output function written in Eq. (2.8).

5.2 Method

The method consists of a graph generation algorithm tailored to the production of small organic molecules, and its implementation with GNNs. In particular, we focus on unconditional generation, meaning that the compounds are generated from scratch, without forcing particular properties. The procedure is sequential, and can be divided in steps: Each graph is generated one node at a time.

5.2.1 Generative Algorithm

The generation of a labeled graph $G = (V, E)$ is handled as a sequential process, starting with an initially empty E and with a single vertex $V = \{v_0\}$. The label l_0 of v_0 is sampled from a distribution of labels D_0, which is learned from the training set. Each step consists in adding a new node to the graph and connecting it to the other nodes. The algorithm focuses on one node v_i at a time, generating all its neighbors before focusing on the following node: $i = i + 1$. This process will be referred to as node expansion. Nodes are indexed according to the order in which they have been generated, so that, for instance, the third generated node v_3 will always be the fourth node to be expanded (v_0 is the first). The process stops when all the nodes have been expanded ($i > |V|$) or when the maximum number of nodes has been reached ($|V| = |V_{max}|$).
As a new node v_j is generated, first it is connected to the node v_i which is being expanded, then it can be linked to the other vertices $V \setminus \{v_i, v_j\}$. While the set of edges generated in the latter phase can be empty, the (v_i, v_j) edge is always generated. This constraint ensures that the generated graph is always connected, without impairing generality: any graph can still be produced.
We can define three problems that must be solved to carry out a generative

step. Each problem corresponds to a function the model will provide: node generation ($P1$), first edge classification ($P2$), additional node linking ($P3$).

- $P1$ decides whether to expand v_i with a new neighbor node v_j or to stop its expansion. If v_j is generated, $P1$ also returns its label l_j.
- $P2$ is called after a new node v_j has been generated. It determines the label $e_{i,j}$ of the edge (v_i, v_j).
- $P3$ is called after a new node v_j has been generated and connected to v_i. It determines the existence of any possible edge connecting v_j to any other vertex $v_k \in V \setminus \{v_i, v_j\}$. The labels of all the generated edges are also returned. All the edges are processed in parallel. The main drawback of this approach is that the dependencies between edges are ignored, but it also brings the advantages of avoiding edge ordering biases and of significantly reducing the time cost.

The generation algorithm is summarized in Algorithm 1.

5.2.2 Implementation with Graph Neural Networks

Each of the functions $P1$, $P2$, $P3$ described in Subsection 5.2.1 is implemented by a dedicated GNN module, which will be referred to as M1, M2, M3, respectively. Each of the modules is trained separately, and one step at a time, assuming the other two modules' decisions to always correspond to the ground truth. This is a strong assumption, which will prevent the model from exploring possible different solutions, but it dramatically simplifies the training procedure. Another advantage of this paradigm is the fact that, each being trained separately from the others, the modules can be recombined to build new versions of the model. If a module needs to be optimized there is no need of re–training the other two.
In order to generate labeled graphs, we need to make some assumptions on the nature of node and edge labels. Three main cases can be identified: unlabeled graphs, graphs with continuous node and edge labels, graphs with a finite set of node and edge types. In this Chapter, we will focus on the third case, which corresponds to the typical setting in molecule generation problems. Thus, in the following, we assume that the label l_i of any node v_i belongs to a finite set $l_i \in T_v$, the label $e_{i,j}$ of any edge (v_i, v_j) belongs to a finite set of types $e_{i,j} \in T_e$, and T_v and T_e are defined by the dataset.

Algorithm 1 Graph generation algorithm.

procedure GENERATE($G = (V, E)$)
  $V \leftarrow \{v_0\}, l_0 \sim D_0$
  $E \leftarrow \emptyset$
  $i \leftarrow 0$
  $j \leftarrow 1$
  while $(i < |V|) \wedge (|V| \leq |V|_{max})$ **do**
    $gd \leftarrow P1(V, E, i)$
    while $gd \neq stop$ **do**
      $V \leftarrow V \cup \{v_j\}, l_j \leftarrow gd$
      $E \leftarrow E \cup \{(v_i, v_j)\}$
      $e_{i,j} \leftarrow P2(V, E, i, j)$
      for $k \in [0, j-1], k \neq i$ **do** ▷ Parallel Execution
        $ld \leftarrow P3(V, E, k, j)$
        if $ld \neq disconnected$ **then**
          $E \leftarrow E \cup \{(v_k, v_j)\}$
          $e_{k,j} \leftarrow ld$
        end if
      end for
      $j \leftarrow j + 1$
      $gd \leftarrow GeneratorDecision(V, E, i)$
    end while
    $i \leftarrow i + 1$
  end while
  return $G = (V, E)$
end procedure

The GNN modules generate nodes and edges along with their labels. With reference to Algorithm 1, the following holds.

- M1 faces a node–based classification problem, as it decides whether to stop the expansion of the current node v_i or to generate another neighbor v_j, and, in case, which label to assign to v_j. The set of output vertices of M1 consists only of v_i: $V_{out} = \{v_i\}$. The output classes correspond to the union of the *stop* decision to the set of vertex types $\{stop\} \cup T_v$.

- M2 deals with an edge–based classification problem, since it generates the label of the edge connecting the node being expanded v_i and its new neighbor v_j. The set of output edges of M2 consists only of this edge $E_{out} = \{(v_i, v_j)\}$. The output classes correspond to the set of edge types T_e.

- M3 works on an edge–based classification problem, since it predicts the existence, and, in case, the label, of every possible edge connecting the new node v_j to the other nodes in the graph, except the node being expanded v_i. These calls are carried out in parallel and integrated in a single prediction from M3. This idea has the drawback of considering each predictable edge as if it were independent from the other predictable edges, but it also allows to avoid the biases introduced by taking the decisions in sequence and it speeds up the procedure. To do so, the graph G is extended with a set of provisional edges $E_p = \{(v_k, v_j) : v_k \in V \setminus \{v_i, v_j\}\}$. The module M3 takes in input the new graph $G' = (V, E') : E' = E \cup E_p$. The set of output edges for M3 is $E'_{out} = E_p$. The output classes correspond to the union of the *disconnected* decision to the set of edge types $\{disconnected\} \cup T_e$.

An example step of this algorithm is visually summarized as a flowchart in Fig. 5.1.
To learn a stochastic behavior from the supervisions, which are samples from a categorical distribution, we resorted to a Gumbel softmax output layer [143], based on the Gumbel–Max method for sampling from discrete distributions [144] [145]. This approach allows to backpropagate through an arbitrarily close approximation of the categorical distribution. The softmax can be annealed, by decreasing a temperature parameter τ, from a less accurate approximation (which tends to a uniform distribution for $\tau \to \infty$)

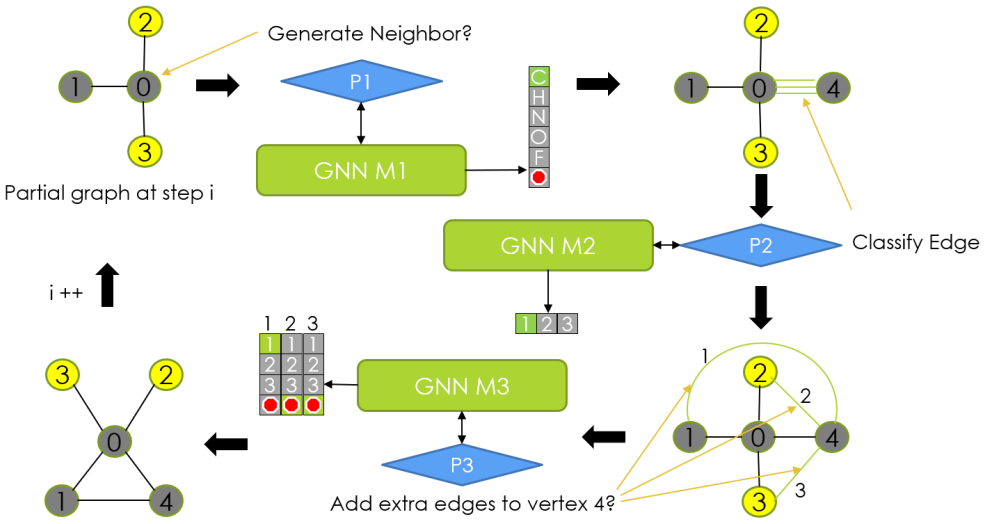


Figure 5.1: Flowchart of the generation algorithm. An example step is summarized, with the three GNN modules (M1, M2, M3), the three problems they are assigned to (P1, P2, P3), their inputs and their outputs. Grey nodes represent carbon atoms, while yellow nodes represent hydrogen atoms. Green edges stand for candidate edges, while black edges represent single bonds. C,H,N,O,F are the element symbols. Classes 1,2,3 represent single, double, and triple bonds, respectively. Red octagons stand for the *stop* decision (M1) or the *do not generate this edge* decision (M3)

to a closer approximation (which tends to the discrete distribution itself for $\tau \to 0$). Lower temperatures come at the cost of an increasing gradient variance. The choice of two parameters τ_{max} and τ_{min}, and a curve, will determine the annealing path. Annealing while training has the positive effect of encouraging the exploration of alternatives to the decision with the highest estimated probability in the early phases, to then converge to more canonical decisions in the final training epochs, when the estimation of the class probabilities has gained higher reliability. This is very important to prevent the networks from learning repetitive patterns, and to avoid mode collapse (i.e. generating always nodes of the same type, based on the highest prior probability).

5.2.3 Graph Preprocessing

To build the training, validation, and test sets for M1, M2, M3, the molecules from the dataset under analysis are pre–processed. For each generative step,

we need an input graph, the index of the focus node, and a supervision. Each molecular graph $G = (V, E)$ is decomposed in a sequence of incomplete graphs, one for each generative step.
For M1, the sequence is composed of $n = 2|V| - 1$ graphs. The first graph contains only one node $G_0 = (V_0 = \{v_0\}, E_0 = \{\})$, any intermediate graph $G_i = (V_i, E_i)$ corresponds to an incomplete subgraph of G, $G_i = (V_i \subset V, E_i \subset E)$, and the last graph is complete $G_{n-1} = G$. For M2 and M3, the sequences are composed of $n = |V| - 1$ graphs, because M2 and M3 are not called after the $|V|$ *stop* decisions from M1 (see Algorithm 1). The graphs $G_i = (V_i \subset V, E_i \subset E)$ acquire nodes and edges as i grows.
The sets are built so that graphs from the same generative sequence (which correspond to different levels of completion of the same original graph) belong to the same set (and to the same batch). This is particularly important to avoid evaluation biases deriving from testing or validating on examples which have slightly different replicas in the training set.

5.2.4 Node Ordering

To define the generative sequences of the graphs, a node ordering needs to be established. This will determine the order in which the nodes of each graph must be generated, and, consequently, the sequences of input graphs and supervisions described in Subsection 5.2.3. The model is expected to learn this generative strategy from the training set, so that, for instance, a training set in which carbon atoms have higher priority will teach the model to generate carbon neighbors first. Theoretically, being V a set with no given ordering, the model would benefit from being trained on any possible node ordering. Since this is impossible from a computational point of view, some constraints must be imposed to reduce the number of orderings from $o(|V|!)$ to a computationally feasible level. In this work we chose a Breadth First Search (BFS) strategy, which has the additional benefit of reducing the number of link predictions needed at each step [105]. Among the nodes at the same depth level in the BFS tree, node types with lower average centrality are expanded first. The average centrality of node types is measured on the training set, according to the Freeman Betweenness Centrality [146]. This boosts both positive effects of the BFS strategy. To further reduce the number of possible orderings of a factor $|V|$, we decided to always start from the same node, which is node 0 of the original node numbering taken from the

dataset. The other nodes are then re–numbered according to the previous rules, making a random choice in any case in which multiple permutations are still possible. The latter two assumptions allow us to retain one unique ordering, coming at the cost of a loss of generality. Although this cost would likely be critical for a truly recurrent model, it is sustainable in this learning framework, in which the correlation between two steps is limited to the output of the first shaping the input of the second. The only input to the model, in fact, is represented by the graph itself, regardless to the many possible sequences of steps that may have brought to its current shape.

5.3 Experiments and Results

A series of experiments were performed, testing MG^{2}N^2 on the QM9 [147], and Zinc [148] datasets, two common benchmarks for the generation of graphs representing small organic molecules, which are introduced in Subsection 5.3.1. The results of this experimentation are then discussed, as they reveal interesting insights into the capabilities of GNNs as molecular graph generators.

5.3.1 Dataset Description

To evaluate MG^{2}N^2, a set of experiments were run on the Quantum Machine 9 (QM9) dataset [147], which is a subset of GDB–17, a chemical universe of 166 billion molecules [149]. QM9 is an ideal benchmark for a new generative model for molecular graphs, as most competitive methods in this area have been tested on this dataset. It is composed of 133,885 compounds, made of up to 9 heavy atoms (C,O,N,F), plus the hydrogens which are bound to them, for a maximum size of 29 atoms. Each molecule is represented as an undirected graph, in which each node corresponds to an atom and each edge corresponds to a chemical bond. The label of each node represents the corresponding atom type, through one–hot encoding, so that $|T_v| = 5$. The label of each edge represents, through one–hot encoding, the type of chemical bond, which can be either single, double or triple, so that $|T_e| = 3$. The output of the modules M1, M2, and M3, defined in Subsection 5.2.2 have dimensions, respectively $|\{stop\} \cup T_v| = 6$, $|T_e| = 3$, and $|\{disconnected\} \cup T_e| = 4$. A random splitting procedure is applied to the dataset, in order to obtain a training set, a test set, and a validation set, composed of 120,000, 10,000 and

3,885 molecular graphs, respectively. The validation set is used, during the training phase, to evaluate the performance of our models on data that are not provided to them directly. The held–out test set allows us to compare the statistics of our sets of generated graphs to the statistics of ground–truth graphs which have never been seen by our generative model, assessing the capability of the model of reproducing the chemical characteristics of QM9 compounds.
Inside each graph, the nodes are re–numbered according to the procedure described in Subsection 5.2.4. To determine the order among the neighbors $Ne(v_i)$ of a generic $v_i \in V$, the average Freeman Betweenness Centrality is measured on the 120,000 training graphs, obtaining the following values: FBC(Hydrogen) = 0.0, FBC(Fluorine) = 0.0, FBC(Oxygen) = 0.115, FBC(Nitrogen) = 0.246, FBC(Carbon) = 0.382.
For a further assessment of the generative performance of the model, a second set of experiments is carried out on the Zinc dataset [148]. This is composed of 249,455 organic molecules of up to 38 heavy atoms (C,O,N,F,P,S,Cl,I,Br). Ring bonds are explicitly labeled as aromatic when part of an aromatic ring. As a consequence, in this setup, we have $|T_v| = 10$, and $|T_e| = 4$. The dataset is split into a training set, a test set, and a validation set of 230,000, 10,000, 9,455 molecular graphs, respectively. The training/validation/test procedure is the same described for QM9. The nodes in each single molecular graph are also re–numbered with the same algorithm.

5.3.2 Experimental Setup

The code for training the GNNs and generating graphs [1] was implemented using Tensorflow [141]. The experiments on QM9, were carried out in the following setup. All the training runs of module M1 were issued on a Nvidia Tesla–V100 GPU, with 32 GB dedicated memory. Training runs of modules M2 and M3 always took place on a Nvidia 2080–Ti GPU. The training set was randomly split in 20 batches of 6,000 graphs each, to reduce the memory requirements. All the experiments used the same split. During the generation of new graphs, even though all the three modules are kept in memory, far less computational resources are needed. The generation sessions were run on the Nvidia 2080–Ti GPU, but required only 0.5 GB of memory. The experiments on Zinc were run on two Nvidia Titan–RTX GPUs, each with 24

[1] Code available at: https://github.com/PietroMSB/MG2N2

GB dedicated memory. The training set was randomly split into 100 batches of 2,300 graphs each to fit in memory.
Table 5.1 shows the configurations of the modules M1, M2, M3 used in the QM9 experiments, which include the neighbor aggregation function, the training epochs, the initial learning rate, the maximum number of iterations for state convergence, and the number of hidden units of the state network and the output network. Each GNN module is composed of a state and an output network. The former is a two–layered MLP implementing the state updating function described in Eq. (2.6). The latter is another two–layered MLP, implementing Eq. (2.7) in M1, and Eq. (2.8) in M2 and M3. The initial values M1(I), M2(I), and M3(I) in Table 5.1 were obtained through a preliminary experimentation, with the goal of maximizing the accuracy of the modules M1, M2, M3, each one independently from the others, on the validation set. Just as if the modules had been classifiers, accuracy was calculated as the percentage of correct outputs, according to the single step supervision, and regardless of molecule validity.
For the Gumbel softmax annealing path, based on [143], we initially chose a linear descent from $\tau_{max} = 5.0$ to $\tau_{min} = 1.0$ during training. Tests on different linear configurations did not bring improvements. In particular, annealing to temperatures $\tau < 1.0$ brought the model to an unwanted repetitive behavior. Therefore, we kept the initial annealing path for all the successive training runs. All the models were trained with an Adam optimizer [54] and cross–entropy loss, which does not require adjustments to work with the Gumbel softmax output layers.

5.3.3 Evaluation

The evaluation of generation performance is twofold. On the one hand, the metrics for unconditional generation introduced in [113] are used to measure the validity, uniqueness and novelty of the generated graphs. On the other hand, the distributions of the chemical properties of the compounds can be compared to those measured on the test set, assessing the model's capability of reproducing the characteristics of QM9 compounds. Both evaluations are carried out on batches of 10,000 generated graphs.
Let Gen be the set[2] of generated compounds, $Val \subseteq Gen$ be the subset

[2]More precisely, here we are using the multiset, an extension of the standard set which can contain multiple copies of the same instances.

Module	Aggregation	Epochs	LR	k_{max}	HU_{state}	HU_{out}
M1(I)	sum	700	4×10^{-3}	5	30	50
M1(II)	sum	1500	2×10^{-3}	6	100	60
M1(III)	sum	2000	1×10^{-5}	6	100	60
M2(I)	avg	500	2×10^{-3}	3	20	50
M2(II)	avg	1000	1×10^{-3}	4	40	60
M3(I)	avg	500	2×10^{-3}	6	20	50
M3(II)	sum	500	2×10^{-3}	6	20	50
M3(III)*	avg	500	2×10^{-3}	6	20	50
M3(IV)*	sum	500	2×10^{-3}	6	20	50

Table 5.1: Different module configurations for QM9 are identified by the module number M1, M2, or M3 introduced in Subsection 5.2.2, and by a sequential version number (I, II, ...). Hyperparameters correspond to: neighbor aggregation function (Aggregation), training epochs (Epochs), initial learning rate (LR), maximum state convergence iterations (k_{max}), hidden units of the state network (HU_{state}), and hidden units of the output network (HU_{out}). M3 versions marked with * were trained with class weights to balance the supervisions.

of chemically valid compounds, and $QM9$ be the set of molecules in the dataset. Validity is calculated as the fraction of chemically valid molecules over the total generated molecules: $Validity = |Val|/|Gen|$. Uniqueness is the fraction of unique molecules among the valid ones: $Uniqueness = |uniq(Val)|/|Val|$, where $uniq$ is a function that takes in input a multiset and returns the corresponding set, from which the duplicates are removed. Novelty is the fraction of unique molecules which do not match any QM9 compound: $Novelty = (|uniq(Val)| - |uniq(Val) \cap QM9|)/|uniq(Val)|$. We also define an additional measure, that combines the three previous metrics and accounts for the fraction of valid, unique and novel molecules over the total generated ones: $VUN = Validity \times Uniqueness \times Novelty$ [3].
The chemical properties include the molecular weight of each compound, the logarithmic octanol/water partition coefficient (logP) [150], and the quantitative estimate of drug–likeness (QED) score [151]. The logP coefficient quantifies the solubility of a molecule in polar or non–polar solvents, while the QED score assesses the drug–likeness of a compound, summarizing in a single measure the following chemical descriptors: polar surface area, molecular weight, logP, number of rotatable bonds, numbers of hydrogen bond

[3]The goals of optimizing validity, uniqueness or novelty are usually in contrast with each other. For instance, improving novelty often comes at the cost of decreasing validity. For this reason, we decided to introduce the new metric VUN, which, by combining the three measures, may provide a more global view on the performance of a model.

donors and acceptors, number of aromatic rings, potential structural issues. The validity as well as the chemical properties of each compound are assessed with the RDKit package [4]. In order to determine the uniqueness and novelty of a molecule, we resorted to the graph isomorphism function of the NetworkX package [152].

5.3.4 Results and Discussion

The first experiment, which was carried out on the QM9 dataset, consisted in a study of the role played in the algorithm by the hyperparameter $|V|_{max}$, which controls the maximum number of nodes in a generated graph. In principle, our model, being trained step by step, can extend its generation procedure for an arbitrary number of iterations, until it stops itself on a complete molecule. This feature could be exploited to extend the generation domain to molecules which are larger than those seen during training, while retaining the same generic patterns. Using M1(I), M2(I), and M3(I), defined in Table 5.1, we explored different thresholds for the maximum number of generated nodes $|V|_{max}$. The natural value for this dataset is $|V|_{max} = 29$, which corresponds to the largest graph size in the training set. As described in Subsection 5.2.1, the generation procedure stops when the number of vertices reaches $|V|_{max}$. This means that any graph still incomplete at that point will not correspond to a valid molecule. Intuitively, raising $|V|_{max}$ will increase the amount of valid generated compounds. Even if this is confirmed by the results reported in Table 5.2, the additional valid molecules, being heavier than average, alter the property distributions of the batch. Moreover, as shown in Fig. 5.2, their QED is below average. Falling in a region of low to very low drug–likeness, these compounds are not useful in the scope of generating new potential drugs. These considerations suggested to keep $|V|_{max} = 29$ for the subsequent experiments.

Starting from the baseline configuration C1 (see Table 5.4), in which the modules are optimized separately and not on the evaluation metrics chosen for our task, we explored the hyperparameter space in search of a better configuration[5]. The first step consisted in increasing k_{max} and the number

[4] RDKit: Open–Source Cheminformatics Software, by Greg Landrum. URL: https://www.rdkit.org/

[5] A systematic search on a grid of configurations was computationally infeasible. Moreover, since the generative models are evaluated with antagonist metrics, it is impossible to

Max size	Validity	Uniqueness	Avg. QED	Avg. Mol. Wt.
29	0.491	0.813	**0.448**	**124.6**
40	0.593	0.845	0.438	144.7
80	0.688	0.866	0.408	172.9
1000	**0.781**	**0.879**	0.366	231.3

Table 5.2: Higher values of $|V|_{max}$ on generation batches from the same model setup produce more valid and unique compounds. The divergence of average QED and molecular weight from the values taken on the validation set (0.478 and 127.5, respectively), however, suggests that the best configuration is $|V|_{max} = 29$.

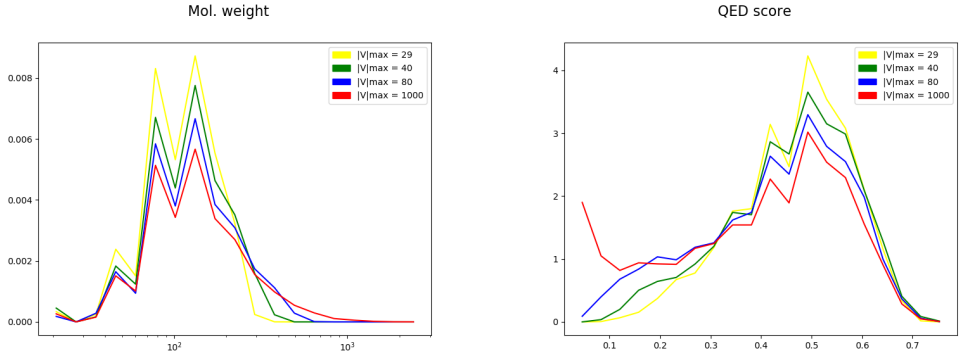


Figure 5.2: Logarithm of the molecular weight (left) and QED (right) distributions of generated graphs with different values of $|V|_{max}$. It can be observed how higher thresholds imply the generation of heavier compounds, with lower QED.

of hidden units in the first two modules, in order for them to better capture complex molecular patterns during training. Using this new configuration (M1(II) and M2(II)), we explored different setups for M3. In particular, to establish the ideal neighborhood aggregation method, M3(I) was compared to M3(II). Then, the same comparison was repeated on M3(III) and M3(IV), which were trained with class weighted supervisions[6]. This latter measure was meant to decrease the learning bias from the very unbalanced prior class probability distribution, which could prevent the model from learning the chemical rules (roughly, 97% of the examples belong to the *disconnected*

optimize the configuration for all of them. Thus, we have heuristically selected the most promising solutions and reported those experiments which, in our opinion, are the most interesting.

[6]The error on each pattern is multiplied by the inverse of the prior of its target class. In this way, the GNN will tend to produce a balanced output over all the classes. At test time, the output is re–multiplied by the vector of prior class probabilities, to restore this important piece of information.

M3 Module	M3 Agg.	M3 Wts.	Validity	Uniqueness	Avg. QED	Avg. Mol. Wt.
M3(I)	avg	no	**0.511**	0.888	**0.461**	134.8
M3(II)	sum	no	0.507	0.887	0.460	135.1
M3(III)	avg	yes	0.476	**0.892**	0.459	**134.2**
M3(IV)	sum	yes	0.499	0.888	0.460	134.3

Table 5.3: Alternative setups for M3 on QM9. Balancing weights bring no advantage on model performance. The two aggregation functions show equivalent results.

Config.	M1	M2	M3	Validity	Uniqueness	Avg. QED	Avg. Mol. Wt.
C1	M1(I)	M2(I)	M3(I)	0.491	0.813	0.448	**124.6**
C2	M1(II)	M2(II)	M3(I)	0.511	**0.888**	**0.461**	134.8
C3	M1(III)	M2(II)	M3(II)	**0.668**	0.340	0.404	75.3

Table 5.4: Summary of the best configurations determined by preliminary experiments on QM9. C3 produces more valid molecules, while the highest QED is obtained by C2. C1 has the closest average molecular weight to the validation set reference (127.5).

class, while the other 3% are distributed over the three bond types). The results of these configurations can be observed in Table 5.3.
This balancing strategy for module M3 did not bring advantages, as it is shown in Table 5.3. This suggests that the GNN can deal with the unbalanced distribution, and efforts to improve the generation performance should focus on other parameters. Besides, the two neighbor aggregation methods appear to be equivalent. A new version of the node generation module, M1(III) was also trained, increasing the number of training epochs and decreasing the initial learning rate (see Table 5.1), in order to stabilize the learning process and avoid early suboptimal solutions. The relevant setups of our model, produced in these experiments, are summarized in Table 5.4.
Table 5.5 compares the results achieved by the most interesting configurations of the proposed MG^2N^2 to various baselines, including the the state of the art for unconditional generation on QM9 (see Subsection 5.3.3 for the metrics). In particular, we compared to: ChemVAE [94], which is based on SMILES strings, and represents a good baseline which does not exploit a graph representation; GrammarVAE [110], which is also based on SMILES, and exploits the grammar of this string representation of molecules; MolGAN [117], which is the best sequential model on this dataset; GraphVAE [113], which is a very competitive (VAE based) method; and MPGVAE [153], a VAE approach in which both the encoder and the decoder are Message Passing Neural Networks [64]. The average values and standard deviations of the chemical descriptors are compared to the equivalent measures from the test

set. As for the MolGAN approach [117], our model does not include the computation of likelihood, nor is it optimized for the global reconstruction of the training examples, as VAEs do [7]. The lack of an explicit global reconstruction penalty is one of the reasons for the very high novelty of the material produced by MG^2N^2: the model is not forced to perfectly reconstruct the molecules on a global basis, but it is forced to correctly reconstruct the local parts of the graph. This approach is expected to preserve a certain degree of validity while encouraging the model to explore more different molecular patterns. Though GraphVAE and MolGAN have higher validity, our model outperforms both of them in terms of uniqueness of the compounds. MPGVAE almost reaches the validity shown by MolGAN, while also achieving good uniqueness, and novelty, and outperforming the other approaches. This advantage is mainly due to the message passing steps performed on the graph in the encoding/decoding pipeline. The aggregated VUN score shows that MG^2N^2 generates the highest percentage of molecules which are valid, unique, and novel at the same time. Notice that, differently to all of the baselines, our method explicitly generates the hydrogen atoms, and all of the hydrogens are required to have been explicitly generated to mark a molecule as valid. This difference is one of the factors determining the lower performance of our approach on the validity metric.

To further assess the chemical similarity between the generated material and the test set of molecules from QM9, we plotted the distributions of the chemical descriptors, which can be observed in Fig. 5.3. For a qualitative visual comparison, showing the similarity between test set graphs and generated graphs, we extracted some valid molecules at random from each set and plotted their structural formulas with RDKit (see Fig. 5.4).

While achieving an acceptable logP distribution, configuration C3 fails to reproduce the QED distribution of the test set. Configuration C2, instead, generates compounds which have very similar logP and QED distributions with respect to those of the test set. This is due to the further optimization carried out on C3: while achieving the goal of building more valid com-

[7] VAEs learn to reconstruct the training examples as closely as possible. The reconstruction penalty is calculated on a global basis, as the Kullback–Leibler divergence between the example graph and its reconstructed version. As the KL–divergence cannot be directly optimized, due to the presence of intractable terms, VAEs optimize the Evidence Lower BOund (ELBO) of these terms, which provides a valuable method to enforce a good global reconstruction.

Model	Valid	Unique	Novel	VUN	Avg. QED	Avg. logP	Avg. Mol. Wt.
ChemVAE	0.103	0.675	0.900	0.063	-	-	-
MPGVAE	0.910	0.680	0.540	0.334	-	-	-
GrammarVAE	0.602	0.093	0.809	0.045	-	-	-
GraphVAE	0.557	0.760	0.616	0.261	-	-	-
MolGAN	**0.981**	0.104	0.942	0.096	-	-	-
Ours(C2)	0.511	**0.888**	**1.000**	**0.454**	0.461 (0.116)	0.272 (1.336)	134.8 (45.7)
Ours(C3)	0.668	0.340	**1.000**	0.227	0.404 (0.088)	0.238 (1.093)	75.3 (52.8)
Test	-	-	-	-	0.482 (0.096)	0.270 (1.325)	127.3 (7.6)

Table 5.5: Validity, Uniqueness, and Novelty of generated compounds assessing the quality of our models and the baselines on the QM9 dataset. The average values of chemical descriptors (Molecular Weight, logP, and QED) are compared to the same quantities measured over the test set. Standard deviations are reported between parentheses. Metrics for GrammarVAE, ChemVAE, and GraphVAE are taken from the GraphVAE article [113]. The performance of MolGAN [117] and MPGVAE [153] are taken from their respective papers.

pounds, it actually went in contrast with the other objectives of generating unique, novel molecules with QM9–like properties. The learning parameters proved to have a role in determining the properties of the model, as we can see by comparing C2 and C3. C2 can be considered as our best model configuration for QM9.

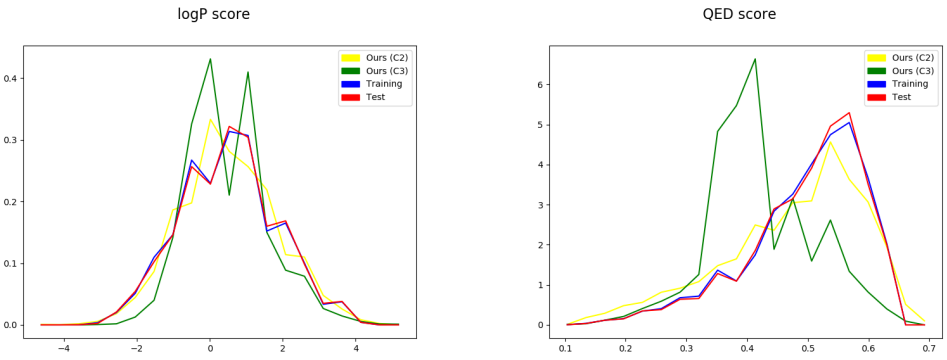


Figure 5.3: logP (left) and QED (right) distributions of generated graphs and training/test molecules. It can be observed how well C2 has generalized the chemical characteristics of the compounds seen during training.

To further assess the performance of our model, a set of experiments was carried out on the Zinc dataset. An optimization procedure analogous to the one described in Subsection 5.3.2 for QM9 allowed to set up the three modules independently. The hyperparameters were then optimized accord-

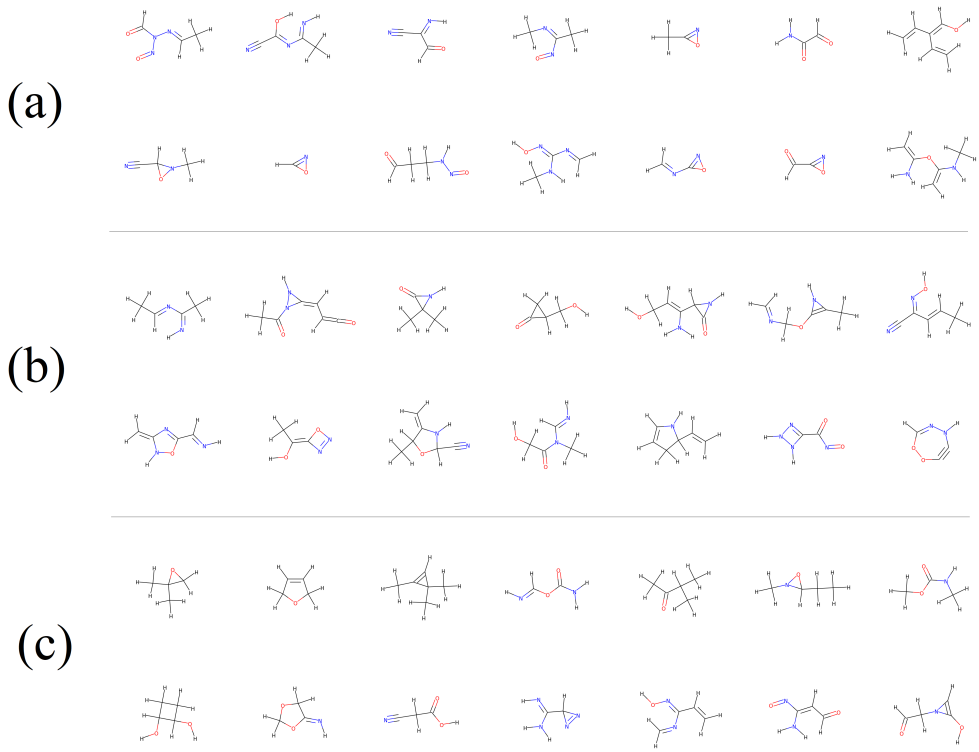


Figure 5.4: Grid representation of random samples of 14 valid molecular graphs generated with configuration C2 (a), 14 valid molecular graphs generated with configuration C3 (b), and 14 molecular graphs from the QM9 test set (c).

ing to the VUN aggregated score. The best model configuration, which was used for the final experiment, is reported in Table 5.6.

To determine the best network parameters for the experiments on Zinc, we started from the best model configuration on QM9, and performed a small grid search in its vicinity, as a more extensive parameter search would have had an infeasible time and computational cost. It can be noticed that modules M1 and M2 required an increased number of parameters to converge, with respect to the QM9 case. This is due to the larger size of the Zinc molecular graphs (up to 38 heavy atoms) compared to the QM9 ones (up to 9 heavy atoms), and to the larger number of node and edge types. The larger size of Zinc molecules also implies a longer generation sequence (on wider graphs), which added to the larger number of examples, and to the larger number of network parameters, multiplies the time and memory burden of each experiment. For this reason, we limited the experimentation

Module	Aggregation	Epochs	LR	k_{max}	HU_{state}	HU_{out}
M1(Zinc)	sum	2000	10^{-3}	6	150	80
M2(Zinc)	avg	1000	10^{-3}	4	50	70
M3(Zinc)	avg	500	2×10^{-3}	6	20	50

Table 5.6: Module configurations used in the Zinc experiment, identified by the module number M1, M2, or M3 introduced in Subsection 5.2.2. Hyperparameters correspond to: neighbor aggregation function (Aggregation), training epochs (Epochs), initial learning rate (LR), maximum state convergence iterations (k_{max}), hidden units of the state network (HU_{state}), and hidden units of the output network (HU_{out}).

Model	Valid	Unique	Novel	VUN
GrammarVAE[110]	0.310	0.108	**1.000**	0.033
ChemVAE[94]	0.170	0.310	0.980	0.052
GraphVAE[113]	0.140	0.316	**1.000**	0.044
CGVAE[154]	**1.000**	**0.998**	**1.000**	**0.998**
Ours	0.753	0.107	**1.000**	0.081

Table 5.7: Validity, Uniqueness, and Novelty of generated compounds assessing the quality of our model and the baselines on the Zinc dataset. The performance of the other models are taken from the CGVAE article [154]

.

on Zinc to the comparison with other models in the literature. Table 5.7 reports the performance of competitive models which were tested for unconditional generation on Zinc, according to the Validity, Uniqueness, and Novelty metrics defined in the GraphVAE paper [113] and to the VUN aggregated score defined in Subsection 5.3.3. In particular, our model is compared to GraphVAE, ChemVAE [94], GrammarVAE [110], and the state of the art approach CGVAE [154]. The great difference in performance between CGVAE and the other methods is largely justified by the different approach to the problem. In CGVAE, molecules are kekulized during the preprocessing stage, thus aromatic bonds are reduced to either single or double bonds. The other hard chemical laws, like valence rules, that would invalidate the generated molecule if violated, are enforced as hard constraints, preventing the possibility of generating invalid molecules [154]. In all the other reported methods, including ours, these rules are learned by the model. Contrarily to what happended on QM9, in this case our model outperforms the standard VAE baselines thanks to the higher validity. This suggests that, as the number of atom types grows, the importance of generating the atoms sequentially and re–examining the output graph at each step, also grows.

5.4 Conclusions

This Chapter introduced a generative model for molecular graphs: MG^2N^2, and a sequential generation algorithm devised for this purpose. The novelty of the presented approach is represented by the exploitation of the capabilities of GNNs to natively process graph–structured data. This allows to use the graph output of the previous step as the network input, which represents an advantage with respect to other sequential methods, that mainly rely on the sequence of previous decisions, rather than on the graph they produce. The modularity of MG^2N^2 implies an easier, less resource demanding, learning process.

In line with all the other sequential methods, and contrarily to VAEs, the generation process is easily interpretable: the steps in which errors occur, or in which specific atoms and bonds are created, can be readily identified in the generation sequence. This feature is very important as it simplifies any process of improvement or repurposing of the model.

The model was tested on a benchmark generation task over the QM9 dataset. The distributions of the chemical descriptors retraced those measured on the held out test set. The quality of generated graphs proved to be very high, allowing our model to outperform very competitive baselines. The same performance level was observed also on the Zinc dataset, when comparing our model to similar approaches.

Future work on this line of research could focus on generalizing MG^2N^2 or a similar model to other molecular graph generation problems, and on extending the approach to conditional generation. A conditional generation model could be implemented by concatenating a vector of desired properties to the input of each module. The comparison with a completely different approach, like CGVAE, which simplifies the generation problem by enforcing chemical rules as hard constraints, suggests that a constrained, or fragment–based, version of MG^2N^2 could improve the performance on datasets of larger molecules, like Zinc. Moreover, studying a theoretical mathematical formulation of sequential generation could also be an important matter of future research.

Chapter 6

Drug Side–Effect Prediction with Graph Neural Networks

This Chapter describes a predictor of DSEs based on GNNs, trained on a heterogeneous relational dataset integrating multiple data sources, which is the subject of publication [P02].
DSEs represent a common health risk, with an estimated 3.5% of all hospital admissions, and approximately 197,000 annual deaths, in Europe alone, related to adverse drug reactions [155]. Such adverse outcomes turn out to be extremely expensive for public care systems. Drug–related morbidity and mortality are estimated to have cost nearly 177.4 billion in the United States alone in the year 2000 [156]. As prescription drug use is increasing [157], the numbers and costs related to DSEs are also expected to rise. DSEs are a huge problem for pharmaceutical companies, as their occurrence during clinical trials slows down drug discovery processes and prevents many candidate molecules from being selected as commercial drugs [82]. Therefore, predicting DSEs before submitting a molecule to clinical trials is extremely important to avoid health risks for participants and cut drug development costs [120].
Computational prediction methods, and in particular DL methods, are techniques of growing importance in this scope [126]. DSEs are, in fact, triggered by complex biological mechanisms, involving interactions between different entities, such as drug functional groups, proteins, genes, and metabolic processes. As a consequence, an efficient predictor should be capable of processing heterogeneous data, accounting for the relationships among different data types [127]. Current ML methods for DSE prediction have increased

the number and variety of features considered for computing the predictions, but they are still widely based on Euclidean (vectorial) data, whereas the relevant information for DSE prediction is relational in nature. This is a limit, since the relational information must undergo a preprocessing to be transformed into vectors, with an inevitable loss of information. In addition, preprocessing methods usually require re–thinking when new features are added. GNNs, instead, can process relational data directly in graph form, exploiting all the structural information.
In this Chapter, we describe a new method for single–drug side–effect prediction based on GNNs, and a graph dataset built for this task, accounting for drug–gene, drug–drug, and gene–gene relationships. To the best of our knowledge, this is the first ML approach to be able to exploit directly graph structured relational data, for the prediction of single–drug side–effects.
The main contributions discussed in the Chapter are as follows.

- The first contribution corresponds to the construction of a relational dataset for the prediction of DSEs, made with data coming from well–known publicly accessible resources. The dataset is a single heterogeneous graph, in which two types of nodes (drugs and genes) share three types of edges (drug–gene, drug–drug, and gene–gene relationships). Both drug and gene nodes have features, accounting respectively for their chemical properties and for their characteristics and function.

- The second contribution consists of DruGNN, a GNN–based method for the prediction of DSEs on the new dataset we constructed. The prediction is set up as a multi–class multi–label node classification problem (applied only to drug nodes, and not to gene nodes), in which each DSE corresponds to a class. We adopt a mixed inductive–transductive learning scheme [140] that exploits both the features of drugs and genes (induction path) and the information on the side–effects of known drugs (transduction path), in order to predict the side–effects of new drugs. The whole method is flexible, since the graph dataset can be easily extended to include other node features and further relationships without changing the ML framework [142].

- The approach is assessed experimentally, with very promising results, showing a good classification accuracy. The performance of DruGNN are compared to those of similar graph–based models (using the same

inductive–transductive scheme) and to those of a deep MLP that cannot exploit relational information. The usability of DruGNN is discussed, as it can be exploited for the prediction of DSEs of new drugs without retraining.

- Finally, two ablation studies, one over the set of side–effects, and the other over the set of features, show the model robustness and the contribution to learning brought by each single data source.

The rest of this Chapter is organized as follows. Section 6.1 describes the dataset, its construction process, and the data sources. Section 6.2 sketches the GNN–based prediction method, giving implementation details of the GNN model specific of this application in Subsection 6.2.1, explaining the inductive–transductive learning scheme in Subsection 6.2.2, and describing the experimental setup in Subsection 6.2.3. Section 6.3 presents the results obtained, and a discussion on their relevance and meaning: results of the ablation studies are presented in Subsection 6.3.1, the comparison with other models is carried out in Subsection 6.3.2, and Subsection 6.3.3 discusses the expected use of the method. Finally, Section 6.4 draws conclusions on the method and the results.

6.1 Dataset

Computational methods for the prediction of DSEs have mainly relied on Euclidean derived features so far. Even methods, like [119], that do use topological information (i.e. about the metabolic network), compress it into a Euclidean space before processing. Since DSEs are triggered by complex biological phenomena, data for predicting DSEs are heterogeneous and come from multiple sources. Drug protein targets are of key importance, as highlighted by the good results of Sparse Canonical Correlation Analysis between drug targets and DSEs [120]. Chemical drug features play an important role too [124], as well as metabolic data [123]. Combining all these pieces of information, even in Euclidean form, yields the best results when using DL predictors [126]. As a consequence, to build our dataset, we integrated information from all of these sources. The main novelty of the presented approach consists in building a graph with these data, and processing the graph as it is, without forcing data objects into Euclidean vectors of features.

The dataset consists of a single graph, in which each drug, as well as each gene, is mapped to a node. Both drug nodes and gene nodes are described by feature vectors. Edges represent drug–drug relationships, drug–gene interactions, and gene–gene interactions. Side–effect labels are associated to each drug node. These labels will be used, according to the inductive–transductive scheme, as either transductive features for known drugs, or class supervisions for new drugs. A sketch of the graph is provided in Fig. 6.1.

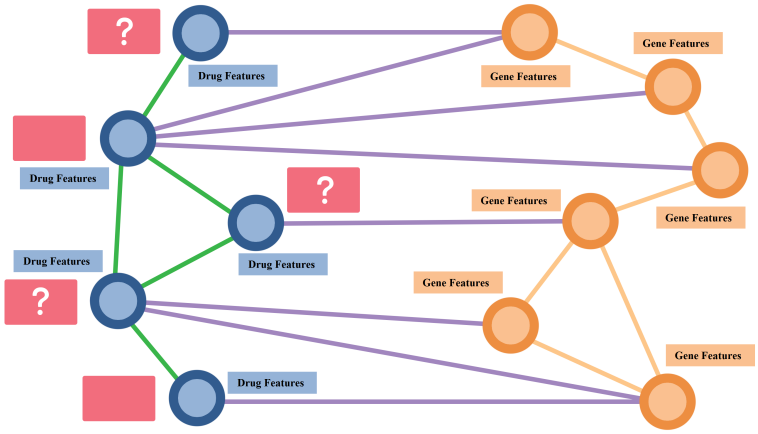


Figure 6.1: Illustration of the graph composition. Drug nodes are represented as blue coloured circles, while gene nodes are represented as orange coloured circles. Blue and orange rectangles account for drug and gene features, respectively. Red rectangles represent classes.

The associations between drugs and side–effects were downloaded from the SIDER database [158], which collects DSE information by aggregating multiple public information sources, summing up to 5,868 side–effects occurring on 1,430 drugs, with a total of 139,756 entries, each accounting for the association of a single drug to a specific side–effect. In our graph, a node was created for each drug. Each side–effect corresponds to a class. Our set of gene nodes, as well as the gene–gene edges, representing the interactions between two genes or their products, were constructed by downloading protein–protein interactions (PPI) information from the Human Reference Interactome (HuRI) [159], and mapping each protein to the gene it is a product of. The product–gene associations were obtained from Biomart [160]. Drug–protein interactions (DPI) were downloaded from the STITCH database [161], one of the most complete and up–to–date DPI databases available. Once again, using Biomart, each protein was mapped to the gene

it is a product of, obtaining the links between drug nodes and gene nodes.
Drug features were retrieved from PubChem [162], which provides seven chemical descriptors for each molecule in the dataset, as well as the SMILES string describing its structure. The seven chemical descriptors consist in: molecular weight (MW), polar surface area (PA), xlogp coefficient (LP), heavy atom count (AC), number of hydrogen bond donors (HD), number of hydrogen bond acceptors (HA), and number of rotatable bonds (RB). In order to better describe each drug molecule, we also translated its SMILES representation to the corresponding structural formula, and extracted its substructure fingerprint, using RDKit software[1]. In order to keep the feature vector size of drug nodes similar to that of gene nodes (gene feature vectors are 140–dimensional and will be described in the following), we opted for drug substructure fingerprints of size 128, bringing the total size of the drug feature vector to 135.
Drug substructure fingerprints were also exploited to build the drug–drug set of edges, accounting for similarity relationships between molecules. In particular, the Tanimoto similarity [163] of the fingerprints of each pair of drugs was measured, adding an edge only to those pairs which were above a similarity threshold (which was set as a hyperparameter at graph construction). Fingerprints were extracted with RDKit again, but with size 2048, in order to better estimate the Tanimoto similarity.
Gene features were obtained from two sources. Biomart [160] provided three pieces of information: the chromosome, which was one–hot encoded for a total of 25 features (22 regular chromosomes, plus X, Y, and mitochondrial DNA); the strand the gene is codified on ($+1$ or -1); the percentage of GC content. Gene ontology [164] provided the molecular function ontology terms each gene is mapped to. These describe the molecular function of each gene, which, combined to the gene–gene interaction links, allow to reconstruct the metabolic network. Since Gene Ontology mapped our genes to a total of 3,422 terms, we clustered the terms to those appearing at the higher levels of the molecular function ontology, using DAVID [165] [166]. This produced 113 unique terms, which were one–hot encoded and concatenated to the gene features obtained from Biomart (for a total of 140 features on each gene node).
We subsequently selected only side–effects with a sufficient number of occur-

[1]RDKit: Open–Source Cheminformatics Software, by Greg Landrum. URL: https://www.rdkit.org/

rences in SIDER: In order for the network to be able to learn the associations of each side–effect, we applied a minimum threshold of 100 occurrences, reducing the number of side–effects in the dataset to 360. After this first filtering step, drugs without side–effects were also removed, reducing the number of drug nodes in the graph to 1,341. Genes with incomplete features were also discarded, along with their gene–gene interactions, bringing the number of gene nodes to 7,881. All the drugs have complete feature vectors, and at least one DPI. DPIs and DSEs of removed drugs were also removed. Drugs are mapped to 360 classes (one for each side–effect), with 96,477 total positive occurrences, and 515,160 negative ones. In this sense, belonging to the positive class for a drug means that it produces the particular side–effect. The target for each drug must be evaluated in relation to every possible side effect (360) since the addressed problem is both multi–class and multi–label. A total of 331,623 edges is present in the final version of the graph: 12,002 gene–gene interaction links, 314,369 DPI, and 5,252 drug–drug similarity links (with a minimum Tanimoto threshold of 0.7). The dataset construction, with all the source databases and preprocessing steps, is sketched in Fig. 6.2.

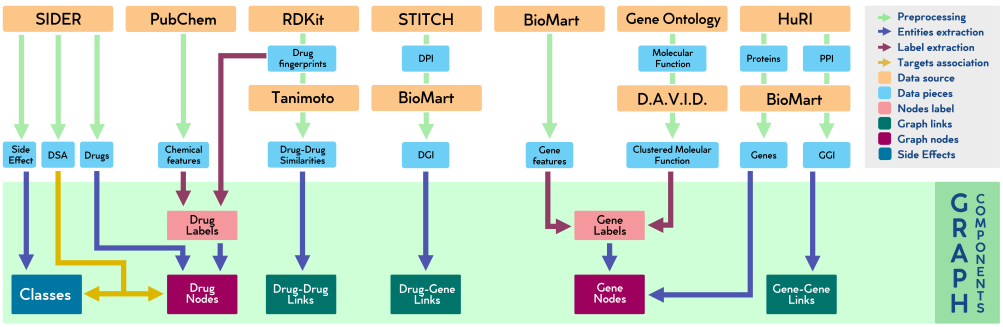


Figure 6.2: Sketch of the dataset construction. Each data source is represented by an orange rectangle. Cyan rectangles represent data pieces. Preprocessing steps are represented by green arrows, which can include feeding data in input to other sources to obtain refined data. Graph node subsets are represented by purple rectangles, with their labels sketched as pink rectangles. Green rectangles are subsets of graph edges, while the blue rectangle represents the classes (side–effects). Red arrows represent the composition of feature labels from data pieces, while blue arrows show the composition of graph entities (nodes, edges, classes). The yellow arrow represents the association of drug nodes to side–effect classes.

6.2 Method

In the work presented in this Chapter, we build a GNN–based DSE predictor called DruGNN. Application specific implementation details of the GNN model are given in Subsection 6.2.1. The final task is to predict the label of drug nodes only, solving a node–based classification problem, with multiple classes (360 side–effects) and in a multi–label setting (each drug can cause multiple side–effects). In order to provide repeatable and comparable results, we set a random dataset split and always use that split throughout the experimentation. The test set contains 10% of the drug nodes and is fed to the network only at test time. In our experiments, we also retain a 10% of the drug nodes as a validation set, in order to check overfitting and stop the training procedure when this occurs. The rest of the nodes (80%) is exploited as a training set.

6.2.1 Model Implementation

This subsection describes the application–specific implementation of the GNN model formulated in Subsection 2.3.1. In particular, in the application presented in this Chapter, the dataset is a heterogeneous graph, in which there are two types of nodes (drugs and genes). Therefore, the GNN model is a Composite GNN, the formulation of which is given in Subsection 2.3.3. The state updating functions are computed by two MLPs, one for each node type. The general formulation given in Eq. (2.10) is specified for this application in Eq. (6.1), where N_d and N_g represents the subsets of drug and gene nodes, respectively. Edge labels are not used in this formulation, as edges are not labeled in our dataset:

$$\begin{aligned} x_n^t &= f_{w,d}(x_n^{t-1},\ l_n,\ a \sum_{m \in Ne(n)} (x_m^{t-1})) \ \text{ if } \ n \in N_d \subset N \\ x_n^t &= f_{w,g}(x_n^{t-1},\ l_n,\ a \sum_{m \in Ne(n)} (x_m^{t-1})) \ \text{ if } \ n \in N_g \subset N \end{aligned} \tag{6.1}$$

During the training phase, the two MLPs will learn two different versions of the state updating function. Namely, f_w^d will be applied to compute the states of all drug nodes $N_d \subset N$, while f_w^g will calculate the states of all gene nodes $N_g \subset N$. The output network is applied to the subset of nodes in the graph for which an output is requested. In our case, these correspond

to the drug nodes. Since this is a problem of node classification the output function is exactly the one described in Eq. (2.7).

6.2.2 Inductive–transductive learning scheme

In the work presented in this Chapter, we make use of a mixed inductive–transductive learning scheme [142], in which the network learns the node–class associations exploiting a double mechanism. In standard inductive learning, the GNN model would predict the side–effects of drugs based on the node features of drugs and genes, and the graph connectivity. In a transductive learning setup, the GNN model would make the predictions based on the known side–effects of other drugs. In our mixed inductive–transductive learning scheme, the GNN model exploits both mechanisms at the same time.

The learning scheme entails splitting the training set into ten batches. The network learns the input–supervision association on one training batch at a time, while the other nine batches are exploited as a transduction set. The features of each drug node in the transduction set are augmented with the transductive features, corresponding to the occurrence of the 360 side effects on that node. When analyzing the validation set, the full training set is exploited, in the same way as described before, as the transduction set. When analyzing the test set, the transduction set is composed of both the validation set and the training set.

This scheme is particularly appropriate for the expected use of our dataset and tool: the idea is to exploit the known DSE associations to predict the DSEs of newly inserted drugs, and the mixed inductive–transductive scheme simulates this behaviour at training, validation, and test times.

6.2.3 Experimental setup

The network hyperparameters were tuned with an extensive grid–search over the validation set. In particular, we analyzed all the hyperparameter values described in Table 6.1 and their combinations. Each element in the grid was analyzed by measuring the average model accuracy in a training/validation experiment with five repetitions.

After tuning the hyperparameters, in order to check the learning capabilities of the DruGNN on the dataset, and in particular the effect on the learn-

Hyperparameter	Values	Best Config.
Activation	relu, selu, tanh, sigmoid	relu
Initial Learning Rate	10^{-2},10^{-3},10^{-4}	10^{-3}
State Dimension	10,50,100,200	50
Hidden Units	100,200,500	200
Neighborhood Aggregation	average, sum	average
Dropout Rate	0.5, 0.3,0.1,0.0	0.0

Table 6.1: Hyperparameter values analyzed during the grid search procedure, and best configuration obtained.

ing process of the reduction or expansion of the set of side–effects, we set up a dedicated series of experiments. In this part of the experimentation, which consists of an ablation study over the set of side–effects, our model was trained and tested on versions of our dataset with progressively reduced numbers of side–effects: only the k most common side–effects were retained, with k assuming values $\{360, 240, 120, 80, 40, 20, 10, 5\}$.
To evaluate the importance of the contributions of the different data sources, another ablation study was carried out. We grouped the features and the edges by source and eliminated one feature/edge group at a time from the dataset, evaluating the performance of the model in absence of that group. The performance gap obtained gives an estimate of the importance of the features that were kept out. There are seven feature/edge groups in our dataset, each of which was analyzed in an experiment repeated five times. Once again, we always used the same dataset split and the same transductive learning scheme described for the previous experimentation (see Subsection 6.2.2).
Eventually, the DruGNN was compared to other competitive GNN models with different characteristics, in order to assess its performance with respect to the alternative solutions. In particular we focused on two powerful models: GCNs [65], which exploit convolutions to aggregate information coming from different locations across the graph, and have shown competitive performance on many different tasks; GraphSAGE [53], which are versatile networks that can be configured with various aggregation and state updating functions, being potentially competitive on every graph dataset. Additionally, we also compared to a simple MLP, in order to assess the difference

between a graph–based model and a Euclidean predictor. In particular, after a small optimization over the validation set, we used a three–layered MLP. It was not possible to include previously published DSE predictors in the comparison, as our dataset is completely novel, and graph–structured, making it impossible to adapt to the feature sets of the predictors available in the literature (see Subsection 3.3). We remind that no graph–based predictor was published for this task so far.

6.3 Results and Discussion

The hyperparameter search described in Subsection 6.2.3 produced a model with an accuracy over the validation set of 87.22%. The same model, evaluated on the held–out test set obtained an accuracy of 86.30%.

6.3.1 Ablation Studies

Given the best model configuration obtained in this first set of experiments, we investigated the contribution of the side–effects to the learning capability of the network. We ranked the side–effects by occurrences, and then we progressively reduced the size of the set of side–effects, by selecting only the most common ones. The average accuracy over five repetitions was measured over the held–out test set. Results are reported in Table 6.2.

DSE	360	240	120	80	40	20	10	5
Acc.%	86.3	81.5	73.2	68.5	63.0	61.8	67.1	74.7
Bal.%	58.1	58.6	60.0	60.5	62.1	59.9	57.7	56.2

Table 6.2: Average accuracy percentage (Acc.%), and average balanced accuracy percentage (Bal.%) obtained on the test set by training and testing the model on progressively smaller sets of side–effects (DSEs).

Since we are dealing with a multi–class multi–label classification task, each class membership can be seen as a problem to be learned independently and in parallel with respect to all the other classes. As a consequence, the first expectation would be that, increasing the number of classes, the network would have to learn a more complex algorithm, needing to solve more problems in parallel. On the contrary, the results reported in Table 6.2 show a

clear tendency of improvement of the performance for larger sets of side–effects. This counter–intuitive behaviour can be ascribed to the network ability of learning intermediate solutions, which are useful for all or large subsets of the classes, with an effect very similar to transfer learning. This can be particularly evident in our system, in which transfer learning between classes is fundamental because of the relatively small dimension of the set of drugs, with the additional bonus of avoiding overfitting. An inversion of this behaviour can be observed at lower set dimensions (up to 20), where transfer learning becomes less easy and convenient and the network learns to treat each class independently.
The unbalanced nature of the problem also plays a role, though. The side–effects with less occurrences are highly unbalanced in favour of the negative class, while the side–effects with more occurrences are unbalanced in favour of the positive class. The balance shift as less common side–effects are removed likely plays an important role in this scope.
A second ablation study was carried out on the feature/edge groups coming from different data sources. The accuracy of the model, trained and tested in absence of the data group, was evaluated and averaged over five repetitions of the same experiment. Since the groups of features are of different sizes, to better weigh the importance of each, we also measured the DPF (Difference Per Feature) score: this is the performance difference with respect to the complete model, divided by the number of features in the group. The description, and the corresponding performance loss observed in the ablation study, of each data group, are reported in Table 6.3.

Table 6.3 shows that each data source has a positive contribution on the GNN learning process. In particular, deleting the drug fingerprints brings the largest performance drop, which can be explained by the importance of the drug substructures in determining the side–effects, but also by the large number of features (128) assigned to this data group. Proportionally, looking at the DPF score, the seven PubChem descriptors have the highest contribution, as it could be expected given their chemical relevance. The gene features also have a relevant impact on performance, with the Biomart derived features having a DPF equal to that of drug fingerprints. Edges also showed to be important, as deleting each edge set leads to a performance drop. Results suggest that drug similarity relations are the less important, likely because drug similarity can be inferred by the network on the basis of the fingerprints and of the drug–gene interactions.

Group	DPI	PPI	DDS	FP	PC	GO	BM
Type	E	E	E	DF	DF	GF	GF
Acc.%	86.14	86.18	86.23	85.81	86.20	86.17	86.20
Diff.	0.16	0.12	0.07	0.49	0.10	0.13	0.10
Count	-	-	-	128	7	113	27
DPF	-	-	-	0.004	0.014	0.001	0.004

Table 6.3: Average accuracy percentage (Acc.%), and difference with respect to the model trained on the complete feature set (Diff.), obtained on the test set by training and testing the model on our dataset in absence of the corresponding feature/edge group (Group). Each group is associated to a type: E (Edges), DF (Drug Features), GF (Gene Features). For DF and GF data groups, the number of features (Count) and the difference per feature (DPF) are reported: the latter is obtained by dividing the difference (Diff.) by the number of features in the group (Count). Our data groups are: DPI (Drug–Protein Interactions), PPI (Protein–Protein Interactions), DDS (Drug–Drug Similarity), FP (FingerPrints), PC (PubChem), GO (Gene Ontology), and BM (BioMart).

Although each group of features and edges has a positive contribution to model performance, the small performance drop obtained by switching them off tells us that the model is robust. In fact, it works almost as well as the complete version even when entire sets of edges or features are deleted. We can therefore hypothesize the following. On the one hand, GNNs are expected to be robust, on the basis of previous systematic ablation studies that demonstrated their capabilities on many types of graph datasets [56]. On the other hand, the large quantity of features and edges, and the heterogeneous nature of our data sources, likely boost the model's robustness.

6.3.2 Comparison with Other Models

To assess the capabilities of DruGNN with respect to other GNN variants, and with respect to non–graph–based Euclidean models, we also made a comparison with GraphSage [53], GCNs [65], and with a simple MLP model trained on a vectorized version of our drug data. The MLP gives a measure of the results that can be achieved by applying a traditional Euclidean predictor on our dataset. The GCN and the GraphSage are trained with the same inductive–transductive scheme as DruGNN. All the models were trained with the binary cross–entropy loss function, Adam optimizer [54], and an initial

learning rate equal to 10^{-4}. A maximum of 500 epochs was allowed for each model, with early stopping on the validation loss, and recovery of the best weights. As expected, all the graph–based models outperformed the standard MLP, showing the advantage given by representing the dataset as relationships on a graph and by learning directly on the graph structure. Moreover, using a GraphSAGE or a GCN approach on this task did not allow to reach the same results we obtained with DruGNN, as shown in Table 6.4. This can be explained by the fact that our model is particularly efficient on node property prediction tasks, as the one presented in this Chapter, while the other GNN models tend to aggregate nodes on a larger scale, getting an advantage on graph property prediction tasks. This is also in line with theoretical studies on GNNs that demonstrate the processing capabilities by simulating the Weisfeiler–Lehman test [56] (see Subsection 2.3.4). Model evaluation is based on the average accuracy percentage obtained over 10 runs of training and testing on the same dataset split.

Model	**Configuration**	**Avg. Acc. %**
DruGNN	$K = 6, SD = 50, DL = 1 \times 200$	86.30%
GCN	$CL = 2 \times 36, DL = 116$	82.94%
GraphSAGE	$CL = 2 \times 72, DL = 1 \times 168$	83.11%
MLP	$DL = 3 \times 25$	77.98%

Table 6.4: Comparison between different models of the GNN family. Model configuration is reported; all of the models were optimized with a small hyperparameter search. K: maximum number of state update iterations for DruGNN; SD: state dimension for DruGNN; DL: number of dense layers and units in each dense layer; CL: number of convolutional layers and units in each convolutional layer. For GCN and GraphSage, the dense layer is the last one before the output layer.

6.3.3 Usability of DruGNN

DruGNN is meant as a tool of real usage, that can help healthcare and pharmacology professionals to predict side–effects of newly discovered drugs or other compounds not yet classified as commercial drugs. The dataset and the software are publicly available on github [2], so that both assets can be

[2]https://github.com/PietroMSB/DrugSideEffects

exploited in further scientific research and by the whole community. Furthermore, both the dataset and the algorithm are scalable: adding new compounds to predict their side–effects does not compromise the network usability (i.e., the network does not need to be retrained from scratch).
An example of such usage is represented by the prediction of the side–effects of Amoxicillin (PubChem CID: 2171), which is part of the held–out test set (and therefore never seen during the training or validation phases). Amoxicillin has been determined to be similar to the following drugs, listed by PubChem CID: 2173, 2349, 2559, 4607, 4730, 4834, 8982, 15232, 22502, 6437075. It also interacts with 76 genes. No other information but the fingerprint and PubChem features of Amoxicillin are available to the model. The network correctly predicts 22 side–effects, among which (listed by SIDER id) quite common and expectable ones, like C0000737 (Abdominal pain) and C0038362 (Stomatitis), but also non–obvious ones, like C0002871 (Anaemia) and C0917801 (Insomnia). It fails to predict 6 side–effects: C0002994 (Angioedema), C0008370 (Cholestasis), C0009319 (Colitis), C0011606 (Dermatitis exfoliative), C0014457 (Eosinophilia), C0036572 (Convulsion). Please notice that Angioedema, Colitis, Dermatitis exfoliative, and Convulsion are indicated as very rare for Amoxicillin. Cholestasis has relatively few occurrences in the dataset, and is therefore difficult to predict. Moreover, the network shows good predictive capabilities on side–effects which are common in the whole drug class Amoxicillin belongs to (represented by the similar compounds in the dataset). In addition, the network predicts only one side–effect which is not associated to Amoxicillin in the supervision: C0035078 (Renal failure).
As shown in the example, to predict the side–effects of a new compound, it is sufficient to retrieve information (coming from wet–lab studies and from the literature) on its interactions with genes, and to know its structural formula. RdKit can be used to calculate the fingerprint, and consequently the similarity to other drugs in the dataset. The PubChem features can either be obtained from a database, or calculated with RdKit. It is then sufficient to insert the compound in the dataset and to predict its side–effects with DruGNN. Visualisation of the DruGNN results and the excepts from the database could then be directly used by doctors and pharmacists.

6.4 Conclusions and Future Work

Combining data from multiple sources is crucial for a DNN to learn complex mechanisms regulating the occurrence of DSEs. In particular, the relational information on the interactions of drugs and genes is well described by a graph structure. Integrating these entities and their relations, we built a graph dataset thought for training and testing graph–based DSE predictors. GNNs showed very good learning capabilities on this dataset, suggesting that a predictor based on this kind of model could help anticipate the occurrence of side–effects. Furthermore, its application on new candidate drugs would help saving time and money in drug discovery studies, also preventing health issues for the participants to the clinical tests.
DruGNN is a modular approach to DSE prediction and is robust to ablation. Moreover, it is easily usable on new drug compounds: it is sufficient to add the new drug, with its features and gene interactions, as a node in the graph, and to run the prediction of its classes. The model does not need retraining, and the same inductive–transductive learning scheme can be used for future additions of compounds and predictions of their side–effects. The prediction relies on a modular multi–omics robust approach, based on information retrieved from publicly available sources. In principle, the same graph could be exploited also to predict the drug–gene interactions of new compounds, by applying link prediction over the gene set.
Since drug structures proved to be one of the most important parts of our dataset for DL, an interesting future direction is represented by the development of a GNN–based predictor that could analyse the structural formulas of the molecules, represented as graphs. These molecular graphs could be augmented with features coming from the gene side and drug–gene relations. In this scope, the algorithm could even be combined with generative models, like MG^2N^2 [11] (described in Chapter 5), that generate molecular graphs of possible drug candidates in large quantities. The task of the DSE predictor would be to screen out all the candidate compounds with high probabilities of occurrence of particular side–effects.
Another very interesting direction is that of specializing the predictor presented in this work, in order to take into account tissue–specific data (i.e. gene expression) and fine–tune a dedicated version of the model for each tissue. This could be made possible by exploiting tissue specific side–effect targets, leading to a more detailed prediction which could also be personal-

ized, given the gene expression values of each individual, as expected in the context of precision medicine.

Chapter 7

Graph Neural Networks for the Prediction of Protein–Protein Interfaces

In this Chapter, a predictor of protein–protein interfaces based on GNNs, corresponding to publication [P11] is presented.
Proteins are fundamental molecules for life. They are involved in any biological process that takes place in living beings, carrying out a huge variety of different tasks. In these molecules, functionality and structural conformation are strictly correlated [167]. Therefore, analyzing structural features of proteins is often useful in understanding which biological processes they are involved in, which ligands they bind to, and which molecular complexes they form.
The structure of a protein can be described at three different levels: the primary structure corresponds to the sequence of amino acids it is composed of; the secondary structure corresponds to the local conformation of the peptide chain, in the shape of α–helices, β–sheets or coils; the tertiary structure represents the three–dimensional configuration of the molecule. Often, two or more molecules bind together to form a protein complex, whose shape goes under the name of quaternary structure. Dimers are the simplest protein complexes, as they are composed of just two monomers.
To form such complexes, monomers interact through specialized parts of their surface, called binding sites or interfaces. These interactions can be studied with the help of graph theory. Indeed, each monomer can be represented as a graph, with nodes corresponding to Secondary Structure Elements (SSEs),

while edges stand for spatial relationships between adjacent SSEs, which can be parallel, anti–parallel or mixed. Using graphs of two different monomers, a correspondence graph can be built, whose nodes describe all the possible pairs of SSEs from the two different subunits [138]. Based on the correspondence graph, identifying binding sites on protein surfaces can be reformulated as a maximum clique search problem [137].
The maximum clique problem is known to be an NP–complete problem, meaning that, except for very small graphs, traditional operations research algorithms [168] will employ a prohibitive amount of time before solving it. From this consideration stemmed the idea of using a ML method to solve the problem with reasonable computational costs. In particular, GNNs [1] look like the perfect model, with their ability to process graph–structured inputs. In this scope, the maximum clique problem consists of a binary classification between the nodes which belong to the maximum clique and those which do not. In particular, the solution proposed in this Chapter entails applying LGNNs [61] to solve the maximum clique problem.
The rest of the Chapter is organized as follows. Section 7.1 illustrates the method, sketching the data acquisition and processing operations in Subsection 7.1.1, giving implementation details on GNNs and LGNNs in Subsection 7.1.2, and describing the experimental methodology in Subsection 7.1.3. Section 7.2 presents and gives interesting insights on the results of the work. Finally, Section 7.3 discusses the results of the approach and draws conclusions.

7.1 Materials and Methods

The method described in this Chapter consists of building a dataset for protein–protein interface prediction, in which pairs of monomers are associated to a correspondence graph. This graph is analyzed with LGNNs in search of the maximum clique.

7.1.1 Dataset Construction

To build the dataset, heterodimers (i.e. dimers formed by two different monomers) characterized by the absence of disulfide bridges, the presence of salt bridges, and protein–protein interaction sites were searched in the Protein DataBank in Europe (PDBePISA) [169]. We obtained a database of

6,695 known proteins for a total of 160,680 monomeric interfaces. To guarantee biological significance, some criteria were enforced: an area of at least 200 Å^2, $\langle x, y, z \rangle$ symmetry, and only two interacting protein molecules. After this operation, we obtained a set of 12,455 interfaces. For every interface, two protein graphs were built, representing two polypeptide chains which interact on the binding site.

The monomeric graphs were built using VPLG [170], with PDB [171] and DSSP [172] files representing the whole protein. Each node v is labeled with a feature vector l_v which consists of: an ID number, the SSE type, the number of occurrences of cysteine and that of the aromatic amino acids (tyrosine, tryptophan and phenylalanine), the percentage of amino acids taking part in the interface and the overall hydrophobicity [173], the charge and Accessible Surface Area (ASA) of the SSE, respectively as the sum of hydropathic indexes, charges and accessible surface areas of each amino acid at pH 7.

Once the graph has been produced for both monomers, it is possible to build the correspondence graph [137, 138]. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be the graphs representing two protein chains and $G = (V_G, E_G)$ be the correspondence graph of G_1 and G_2. Let $v_i, u_i \in V_i$ be two generic nodes in G_i with $i = 1, 2$. Therefore, two nodes $v = (v_1, v_2), u = (u_1, u_2) \in V_G$ are connected by an edge $(v, u) \in E_G$ if and only if $\exists (v_1, u_1) \in E_1$ and $\exists (v_2, u_2) \in E_2$. The edge label $e_{u,v}$ is a one–hot representation of the spatial relationship between two adjacent nodes in G, which depends on the labels e_{v_1,u_1} and e_{v_2,u_2}, so that $e_{v,u}$ is the same edge label if both the edge labels in G_1 and G_2 are equal, *mixed* otherwise. The label of node $v \in V_G$ consists of: an ID number, a one–hot representation of the SSE type, the differences in the occurrences of cysteine and the aromatic amino acids, the arithmetic mean of the two hydrophobicity values, the minimum of the ASAs and the sum of the charges of the two SSEs. In particular, the SSE type of the node $v \in V_G$, which represents $v_1 \in V_1$ and $v_2 \in V_2$, is the same as that of the nodes v_1 and v_2 if both belong to the same SSE class, while it is defined as *mixed* if they belong to different SSE classes.

The node targets were generated with the Bron and Kerbosch algorithm [174], which identified the cliques within each correspondence graph, with a minimum size of three nodes. Subsequently, these cliques were analyzed, in order to determine whether or not they were biologically significant. In this context, a clique is defined as positive or biologically significant if and only if all the nodes belonging to that clique represent pairs of SSEs of different

monomeric graphs, that contain both at least one residue that is part of the interface. Hence, the target attached to each node is a two–dimensional vector containing a one–hot encoding of the two classes: positive if the node belongs to a biologically significant clique, negative otherwise.
We obtained 512 correspondence graphs, each containing at least one biologically significant clique (and any number of negative cliques) composed of three or more nodes. These were not completely connected, often being made of multiple separated connected components. Since many connected components did not include cliques, a pruning strategy was adopted, in order to clean the dataset. The correspondence graphs were split, obtaining a graph for each connected component. We kept only those which contain at least one clique, whether positive or not. This operation produced the final dataset of 1044 connected graphs, 537 of which contain a positive clique, while the remaining 507 contain only negative cliques. Table 7.1 provides numerical information on the dataset, before and after the pruning process.

Dataset	Graphs	Edges	Nodes	Nodes0	Nodes1	%Nodes1
Before Pruning	512	441.203	328.629	325.798	2.831	0.86 %
After Pruning	1.044	274.608	166.424	163.593	2.831	1.7 %

Table 7.1: Dataset statistics before and after data cleaning (Nodes0/1 represent negative/positive nodes)

7.1.2 GNN Implementation

In the work described in this Chapter, GNNs are implemented as formulated in Subsection 2.3.1, with the only difference represented by the fact that edge labels are not taken into account, because edges are unlabeled in this dataset. Moreover, the only neighborhood aggregation function used in this case is the sum. Therefore the hyperparameter a will always have value $a = 1$. The state updating function f_w defined in Eq. (2.6) can be re–written as in Eq. (7.1):

$$x_n^t = f_w(x_n^{t-1},\ l_n,\ \sum_{m \in Ne(n)} (x_m^{t-1},\ l_m)) \tag{7.1}$$

Since this is a node classification task, the output function g_w is implemented exactly as defined in Eq. (2.7).

GNNs are stacked to build up a layered architecture, namely a LGNN [61], in which each layer consists of a GNN. 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 $l_n^i = [l_n, x_n^{i-1}]$ 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. This scheme encourages each layer to refine the solution provided by the previous layers, improving the performance with respect to a single–layered GNN. LGNNs are trained step by step, one layer after the other, from the first to the last. Each layer is trained using the same original targets. Given the operation described for the labels, all the mathematical formulations remain valid for each layer.

7.1.3 Experimental Setup

A binary GNN classifier was developed for the detection of maximum cliques in the correspondence graphs, which addresses the problem as a node–based classification task. Usually, in a classification task, the performance is measured in terms of accuracy. This metric, though, is not precise on unbalanced datasets, like ours. Therefore, we decided to evaluate the model's performance using the F1–Score, which combines precision and recall to provide a balanced measure.

The architecture of the MLP module dedicated to the output function g_w was kept fixed, using a single level MLP and the softmax activation function. On the contrary, a 10–fold cross–validation was performed in order to determine the best hyperparameters for the MLP implementing the state updating function f_w. According to the cross–validation results, the MLP architecture with better performance has got a single hidden–layer with logistic sigmoid activation functions. This setup was used also to test a 5–layered LGNN network, where each GNN layer shares the same architecture.

In order to evaluate the performance of the LGNN, a 10–fold cross–validation was carried out. The LGNN is composed of 5 GNN layers, each with state dimension equal to 3. The state is calculated by a one–layer MLP with logistic sigmoid activations, while the output is calculated with a one–layer

MLP with softmax activation. Since the negative/positive examples ratio is quite large, the weight of positive examples is fixed to the 10% of this ratio, against a weight of 1 for negative examples, in order to balance the learning procedure. The model is trained with the Adam optimizer [54] and the cross–entropy loss function.

7.2 Experimental Results

The best performance is obtained with LGNNs integrating only the state in the node labels. There are slight improvements in precision and more tangible improvements in recall, which gains more than 10 percentage points in the second GNN level, and then continues to grow and stabilize in the following levels, as shown in Fig. 7.1. This architecture is the only one in which we observe a significant increase of the F1–Score, getting more than 6 percentage points from nearly 35% of the first GNN level to more than 40% in the final GNN level, as reported in Table 7.2.
Contrariwise, integrating in the node labels only the output or both the state and the output, the F1–score decreases through the LGNN layers. The other parameters remain almost stable, except for recall, which slightly increases through the LGNN layers. However, the standard deviation of the recall tends to grow, suffering from a marked dependence on the initial conditions of the experiment. The results confirm the expectations based on biological data and show good performance in determining the interaction sites, recognizing on average about 60% of the interacting nodes.

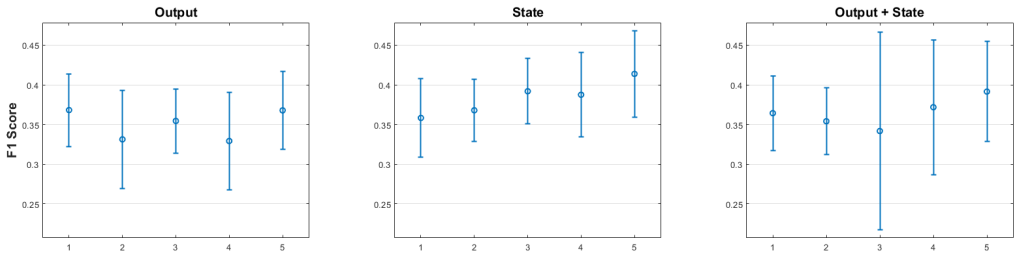


Figure 7.1: 5 levels LGNN 10–fold cross validation results: F1–score

Output	Level 1	Level 2	Level 3	Level 4	Level 5
Precision	0.319(0.069)	0.271(0.058)	0.287(0.049)	0.266(0.046)	0.295(0.07)
Recall	0.455(0.048)	0.447(0.111)	0.476(0.061)	0.446(0.101)	0.517(0.059)
F–Score	0.368(0.046)	0.331(0.062)	0.354(0.04)	0.329(0.062)	0.368(0.049)

State	Level 1	Level 2	Level 3	Level 4	Level 5
Precision	0.31(0.061)	0.279(0.045)	0.322(0.052)	0.295(0.053)	0.328(0.061)
Recall	0.436(0.063)	0.558(0.056)	0.524(0.087)	0.585(0.08)	0.571(0.067)
F–Score	0.358(0.05)	0.368(0.039)	0.392(0.041)	0.387(0.053)	0.414(0.055)

Both	Level 1	Level 2	Level 3	Level 4	Level 5
Precision	0.308(0.063)	0.273(0.052)	0.261(0.106)	0.301(0.064)	0.296(0.06)
Recall	0.46(0.056)	0.544(0.109)	0.52(0.185)	0.518(0.168)	0.597(0.096)
F–Score	0.364(0.047)	0.354(0.042)	0.342(0.125)	0.372(0.085)	0.392(0.063)

Table 7.2: Results obtained with three different LGNN settings: propagating the output, the state or both from one layer to the next

7.3 Conclusions

We addressed the problem of protein–protein interface detection as a search for the maximum clique in the interface correspondence graph. Although the problem is NP–complete, our method, based on GNNs, can find the maximum clique in affordable time. The performance of the model was measured in terms of F1–score and shows that the approach described in this Chapter is very promising, though it can be further improved. One key idea in this direction is that of using graphs in which the nodes correspond to single amino acids, rather than to SSEs. Although this latter approach would increase the complexity of the problem, it would avoid the loss of information we encounter in compressing amino acid features into SSE nodes. Moreover, predictions obtained in this setting would be more accurate, describing the binding site at the amino acid level.

Chapter 8

Other Works

This Chapter provides an overview of the other activities carried out during the Ph.D. and not strictly related to the content of the thesis. Section 8.1, presents the participation into a theoretical activity focused on trustworthy, automatic, interpretable, fair, and secure learning with GNNs [P06]. Section 8.2 describes GlyPipe, a software pipeline for the automatic prediction of glycine mutations that can induce the formation of protein surface pockets, the druggability of which is evaluated with an MLP, based on various types of pocket features. The approach, which is the focus of publication [P07], is described in Subsection 8.2.1, while its applications, corresponding to publications [P08] and [P09], are presented, respectively, in Subsection 8.2.2 and in Subsection 8.2.3. Section 8.3 collects approaches that could help the scientific research on Covid-19. Subsection 8.3.1 introduces a method to predict binding sites on the surface of spike glycoprotein monomers that could host molecules capable of impairing the formation of the quaternary structure of the spike glycoprotein itself. This work lead to publications [P04] and [P15]. Subsection 8.3.2 presents a siamese LSTM for the fast estimation of alignment scores of human coronavirus sequences, and a consequent investigation of similarity between Covid–19 and other human coronaviruses, presented in publication [P10]. Section 8.4, instead, presents an attention–based LSTM predictor of protein secondary structures [P05]. Section 8.5 presents a proof of concept of a mechanism based on GNNs that could facilitate the exchange of experience and support between caregivers of rare disease patients [P14]. Finally, Section 8.6 provides a description of two applications of DL techniques to image analysis tasks coming from the real world: Subsection 8.6.1 describes a CNN model for dragonfly action recognition [P13], while Sub-

section 8.6.2 illustrates a deep CNN approach for the classification of skin lesions [P12],

8.1 Towards Learning Trustworthily, Automatically, and with Guarantees on Graphs

In this Section, the collaboration to a theoretical work, corresponding to publication [P06], is presented. The work is focused on surveying the possibility of learning with GNNs in accordance with a variety of virtuous conditions. These conditions include the possibility of learning in a trustworthy way, and learning automatically. Fairness, security, safety, and robustness are analyzed, as well as privacy and explainability [175]. In particular, the collaborative activity was mainly focused on defining the computational power of GNNs, and surveying the existing methods for this task, as described in Subsection 2.3.4. The unfolding tree method [7] was presented and mathematically formulated. The alternative method [56], based on the Weisfeiler–Lehman isomorphism test [57] was then described, along with the classification of GNN models into tiers of different expressive power according to the possibility of simulating the Weisfeiler–Lehman test. The equivalence between the two methods [59] was discussed and the conditions under which GNNs are universal approximators on graphs were defined.

8.2 GlyPipe: Opening New Protein Surface Pockets

This Section describes GlyPipe: a predictor of mutations that can open new pockets on the surface of protein structures, and the automatic evaluation of the druggability score of the pockets formed with this approach. Two relevant applications [176, 177] of this method are then presented.

8.2.1 Glycine–induced formation and druggability score prediction of protein surface pockets

In this Subsection, a new idea is proposed for the realization of mutated proteins, in order to form new transient pockets on their surface, capable of

hosting drugs. In particular, new allosteric sites are obtained by replacing amino acids with bulky side chains with glycine, Gly, the smallest natural amino acid. We also present a ML approach to evaluate the Druggability Score (DS) of new (or enlarged) pockets. These features are implemented in a software pipeline, called GlyPipe, for the prediction of glycine–mutations that can induce the formation of druggable pockets. This approach is described in publication [P07].
GlyPipe takes protein structure files, in PDB format [171], in input, searches for the best glycine substitution and evaluates the DS of the resulting pocket, returning the mutant structure, the pocket, and its predicted DS. The working steps of GlyPipe are represented in Fig. 8.1.

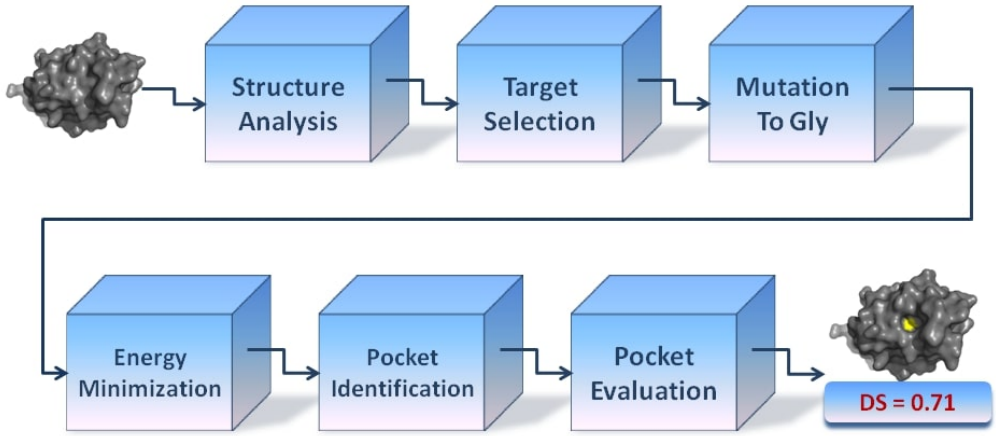


Figure 8.1: Steps of the GlyPipe software pipeline

The first module in the pipe makes use of the Simple Atom Depth Index Calculator [178] and of the POPS software [179] to select residues which are located on the surface, not excessively exposed, and with their side–chain pointing outwards. The set of residues meeting these conditions is sequentially scanned, in search of the amino acid which is most likely to produce a druggable pocket, once substituted with a glycine. The best amino acid types are Phe, Trp, and Tyr, but also other non–polar zero–charge amino acids are considered. The glycine mutation is applied to the protein struc-

ture using PyMOL [1]. The stability of the mutant structure is checked with an energy minimization protocol provided by PyMOL. Once the mutated structure has been minimized, it is analyzed with PockDrug [180]. The new pocket can be detected by comparing the results of this analysis to those obtained on the natural version of the structure. If a new pocket is found in the vicinity of the mutated amino acid, the following step is applied to evaluate its druggability. In some cases, a pocket exists, close to the mutated amino acid, both before and after the mutation: in this case the druggability is evaluated for both versions of the pocket, and compared.

GlyPipe is equipped with an MLP regressor, which takes the pocket descriptors in input and predicts the pocket DS. Relevant descriptors correspond to chemical and physical features of the pocket, such as: hydrophobicity [173], fraction of polar residues, numbers of aromatic and aliphatic residues, charge, diameter, and volume. In order to train and test the model, a dataset of labeled pocket examples was built. We selected 1,200 protein structures, by drawing them at random from a nonredundant list of proteins collected from the PDB [171]. From the set of 1,200 proteins, 10,797 pocket examples were collected. The supervision was provided by evaluating the DS of each pocket with PockDrug. The pocket set was randomly split into a training set (9717 examples) and a test set (1080 examples), with roughly the same ratio of negative to positive examples. After a preliminary hyperparameter search on a validation subset of the training data, the network achieved very good performances in replicating the DS calculated by PockDrug, while being way faster than PockDrug in the estimation, once trained.

An ablation study was also performed, allowing to determine which of the features are more informative. This determined that a subset of 13 features out of 17 allows to obtain the same prediction performances than the full feature set. Moreover, a smaller subset of 8 features reaches slightly less accurate results, while needing much less descriptor calculations. Following this result, an ad–hoc feature extractor was built, that takes in input the pocket file and calculates only those 8 features that are useful for the prediction. This allows much faster DS predictions.

As a final experiment, the pipeline underwent a preliminary test. Sixty structures, none of which belongs to the dataset used for training and testing the DS predictor, were downloaded from the PDB and submitted to GlyPipe. The resulting mutant structures were manually checked and analyzed with

[1] The PyMOL Molecular Graphics System, Version 1.8, Schrödinger, LLC.

PockDrug, obtaining the following results:

- 1 non–druggable pocket closed;
- 9 mutations did not open or close cavities;
- 7 modified pockets (2 of which became druggable);
- 43 new pockets (4 of which druggable).

In all these cases, the DS prediction was in agreement with the DS calculated by PockDrug. We can state that, in this experiment, a new pocket was opened by 71.7% of the mutations. In total, six new druggable pockets were obtained over sixty mutations. Therefore, 10% of the mutations proposed by GlyPipe produced a new druggable pocket.
These preliminary experimental results are really promising, allowing for a preparatory selection of those pockets that actually deserve to undergo a virtual screening procedure in order to find potential ligands, and significantly reducing the computational cost of the whole process. We are not aware of other methods which explore the possibility of creating new protein surface pockets by substituting a large amino acid with a smaller one.

8.2.2 Structural bioinformatics survey on disease inducing missense mutations

In this Subsection, referring to publication [P08], a bioinformatic study on the applicability of GlyPipe (see 8.2.1) is presented.
Understanding the molecular interactions between proteins, nucleic acids and small molecules is fundamental to build reliable methods of genetic medicine [181]. Genotype–phenotype associations in human diseases can be efficiently explored by accessing mutation databases, such as ClinVar [182]. The fact that missense mutations constitute the most common sequence alteration in Mendelian disorders offers a good starting point to understand the mechanisms of disease appearance due to amino acid variations in mutated proteins [183]. Pathogenicity can occur whenever a missense mutation alters the structural stability of the protein and, consequently, its function. The large number of examples of this kind has driven the implementation of several algorithms to predict functional damages caused by amino acid substitutions [184, 185]. Moreover, the abundance of structures that are currently

available in the Protein Data Bank (PDB) [171] allows a detailed view of the protein interactome. The missense mutations provided by ClinVar can therefore undergo a structural analysis, obtaining interface data from the PDBePISA resource [169].
Starting from a set of mutations corresponding to all the ClinVar entries (789,266 as of June 10, 2020, when the study was conducted), we selected only single nucleotide variations that cause protein missense mutations, which had a minimum review status (one star), obtaining 308,326 entries. Excluding entries with intermediate levels of clinical significance, we built our sets of benign (25,579 entries) and pathogenic (21,595 entries) missense mutations. Each mutation was associated with a PDB structure using VarMap [186]. Discarding mutations without structural and/or interface data, we obtained 5,641 benign mutations and 3,018 pathogenic ones. The area around each mutation was analyzed with a method similar to the technique described for GlyPipe (see Subsection 8.2.1).
We investigated the possibility of learning to classify the mutations as either benign or pathogenic with a ML model. We used all the mutations in the benign and pathogenic datasets, which amount to a total of 8,659 examples. Each mutation is described by a 46–element feature vector composed of: the native residue (20–bit one–hot encoding of the amino acid type), the mutant residue (20–bit one–hot encoding of the amino acid type), and a location vector (6–bit one–hot encoding) which indicates where the mutation occurs: core, surface, protein–protein interface, protein–RNA interface, protein–DNA interface, or protein–ligand interface. Reserving 1,000 mutations for testing, we trained three ML binary classifiers on the other 7,659 examples: an MLP, a random forest, and a SVM. After training, the models were tested on the held–out test set of 1,000 mutations, obtaining good accuracy results: 76.0% for the MLP, 75.6% for the random forest, and 74.1% for the SVM.
The experimental results suggest that missense mutations can be evaluated with ML techniques, assessing their pathogenicity based on the wild–type residue, the mutant residue, and the location inside the protein structure. Moreover, a statistical analysis carried out in parallel with the ML study, suggests that arginine and glycine have a special role in this kind of mutation. In particular, the results of the latter analysis clearly indicate that arginine mutations increase pathogenicity whenever they occur at PDBePISA–defined interfaces and, particularly, at protein–DNA interfaces. In the protein–DNA

interfaces, arginine is more than six times more frequent in the benign set than it is in the pathogenic set. This fact is in total agreement with the very critical role that this amino acid has in the interaction with nucleic acids [187]. Moreover, arginine benign mutations are not frequently found in buried protein moieties or in protein–protein interfaces, while glycine benign mutations exhibit the opposite trend. The latter glycine mutations are more frequently found in protein–ligand interfaces, in agreement with the suggested role of this amino acid in stabilizing concave moieties of the protein surface [92]. The prevalent localization of pathogenic glycine mutations indicates that the substitution of glycine with amino acids having larger side chains causes structural stress and, hence, functional changes in the mutated proteins.

8.2.3 Structural bioinformatic survey of protein–small molecule interfaces delineates the role of glycine in surface pocket formation

Understanding the mechanisms of protein–small molecule interaction is a fundamental step to increase our knowledge of life and to establish new therapeutic approaches [188]. In previous studies, described in Subsection 8.2.1 and in Subsection 8.2.2, the role of glycine in interfaces between proteins and ligands, other proteins, and nucleic acids, has been investigated. In this Subsection, describing publication [P09], instead, we investigate the amino acid composition of interfaces between proteins and small molecules, searching for signals that could help designing additional protein binding sites.

To build the dataset on which to carry out this investigation, protein–ligand adducts were derived from the 163,141 structure files that were present in the PDB [171] at the time of the study (April 24, 2020). Files that did not contain exclusively proteins, files containing more than one chain in the asymmetric unit, and files which did not contain ligands were excluded from the dataset, resulting in a set of 45,580 structures. These were filtered in order to eliminate files with 50% redundancy or higher, obtaining a final set of 11,351 structures, which was split into two subsets: 4,947 enzymes and 6,404 non–enzymes. To delineate the amino acid composition of the protein–ligand interface of each structure, we resorted to the PDBePISA interface database [169].

This analysis allowed to count and characterize the occurrences of each amino acid in a large amount of protein–ligand interfaces. This piece of information is of fundamental importance in the scope of using GlyPipe (see Subsection 8.2.1) in order to modify existing protein surface pockets or open new ones that could host ligands. In particular, the possibility of using GlyPipe according to the previously derived amino acid compositions was investigated in a case study on the Hen Egg White Lysozime (HEWL) structure (PDB id: 1LZT). To this aim, GlyPipe was configured to substitute Glycine to the three amino acids with the largest side–chains (Tryptophan, Phenylanine, and Tyrosine). We found that, on the structure under analysis, only Tyr20, Tyr23, Tyr53, Trp111, and Trp123 satisfied all the substitution criteria defined by GlyPipe, and the additional requirement of being distant from the natural binding pockets, in order not to impair the original protein function. As expected, all five mutations produced new deep surface pockets in the respective mutant HEWL structures. Molecular dynamics simulations allowed to establish that none of the mutations altered protein behaviour and functionality, and to verify that the pockets remained opened on a stable basis. The druggability scores of the five pockets were calculated with GlyPipe. All the pockets showed good to very good druggability scores.
The experimental results allowed to draw conclusions on the role of Glycine and other amino acids in protein–ligand interfaces. Moreover, Glycine substitutions, more precisely guided by the information obtained on the composition of such interfaces, can effectively be used to open new ligand–hosting surface pockets in an in silico experiment.

8.3 Structured Data in Covid–19 research

This Section collects two activities on Covid–19 data, carried out in order to help the research against Covid–19 which is focusing the efforts of scientists from a wide variety of different application fields worldwide.

8.3.1 Interfering with Covid–19 Spike Glycoprotein Trimerization

This Subsection discusses a work [189, 190], corresponding to publications [P04] and [P15], on the possibility of impairing Covid–19 Spike glycoprotein (S) trimerization by inserting known drugs into ligand–binding pockets

found in the interface regions of the spike protomers. The S glycoprotein, indeed, is central for COVID–19 infection as it mediates attachment of virions to the host cell receptor, it is involved in cell–to–cell fusion and induces neutralizing antibodies, bearing also virulence determinants [191]. Assembly of protomers into the bioactive form of the trimeric structure of the S glycoprotein has been experimentally proved to be the rate–limiting step for the infecting cycle of the transmissible gastroenteritis coronavirus (TGEV) [192]. The close similarity between S glycoprotein of TGEV and COVID–19, as controlled in the present study, suggests that the same in vitro observations made on the TGEV S glycoprotein assembly [192] should also hold in the COVID–19 case. Therefore, being the trimerization a relatively slow process, this study tries to interfere with it, by finding surface pockets in the interface regions to which ligands can bind, building a physical obstacle to protomer–protomer interaction.

The quaternary structure of the Covid–19 S glycoprotein was downloaded from the PDB [171], with structure PDB ID: 6VSB [193]. After removing all the heteroatoms, depth indices were calculated with SADIC [178], and interface residues obtained from PDBePISA [169]. Using an algorithm based on GlyPipe [92] (see Section 8.2), pockets were located at protomer–protomer interfaces, and their druggability evaluated. In particular, the pockets were selected according to the following criteria: overlapping with interface surface higher than 70%, pocket volume > 300 Å^3, druggability score > 0.7, located in the S2 domain of S glycoprotein, number of pocket residues > 10. Six pockets were found to meet all these requirements. Before selecting possible ligands that can bind to the above–described protomer pockets, the sequence conservation was checked by multiple sequence alignments using Clustal Omega [194]. All residues belonging to COVID–19 S glycoprotein S2 domain, encompassing 660–1273 sequence fragment, exhibited full identity.

After these preliminary results, an investigation was carried out on the existence of small molecules that can interfere with COVID–19 S glycoprotein trimerization through binding to the pockets previously delineated on the protomer surfaces. This task was achieved by downloading the content of DrugBank 5.1.5 [195]. A docking simulation was carried out on the six pockets found in the previous analysis, obtaining a large array of possible ligands. The present analysis of protomer–protomer interfaces of COVID–19 S glycoprotein can be a useful starting point for predicting ligands that are already

in use for other pathologies and, by interfering with quaternary structure assembly of COVID–19 S glycoprotein, can exhibit therapeutic activity against viral life cycle.

8.3.2 A bioinformatic approach to investigate structural and non–structural proteins in human coronaviruses

In this Subsection, the contribution relative to publication [P10] is summarized and briefly discussed. Recent studies confirmed that people unexposed to SARS–CoV2 can show some pre–existing reactivity. The immunological mechanisms underlying this pre–existent reactivity seem to be linked to previous exposure to widely circulating common cold coronaviruses [196], and comes from memory T cells able to specifically recognize a SARS–CoV2 epitope of structural and non–structural proteins as well as the homologous epitope from common cold coronaviruses [196]. Therefore, it is important to understand the SARS–CoV2 cross–reactivity by investigating protein sequence similarities with other circulating coronaviruses. A deeper investigation of cross–reactive T cell immunity to SARS–CoV2 has extensive implications in differential COVID–19 clinical outcomes and can influence the performance of COVID–19 vaccines. Thus, this work can be a starting point for further studies about cross–reactive T cell recognition between circulating common cold coronaviruses and SARS–CoV2.
To make such analysis easier, we implemented a siamese LSTM model for the alignment of Covid–19 protein sequences. To train and validate the model, we built a dataset of examples based on the NCBI [197] protein clusters. The neural network hyperparameters were selected after a grid search. Each Siamese module is composed of a single 32–unit LSTM layer, with input size 550×21. The representations coming from the two modules are combined by a merge layer, followed by a normalization layer. Finally, a single dense layer with ReLU activation estimates the distance between the two elements of the pair. Supervisions are provided by BLAST [198]. Results showed that the siamese LSTM can predict the alignment distance of pairs of proteins with very low error rates, being faster than BLAST once trained.

8.4 A Deep Attention Network for Predicting Amino Acid Signals in the Formation of Alpha–helices

The knowledge of the secondary and tertiary structure of a protein is fundamental for understanding its function and its structure–function relationships. It is now well established that protein structures are mainly determined by their amino acid sequences [199]. Protein folding prediction techniques have been based on this information for decades, but they have not yet reached an acceptable level of accuracy. Methods to experimentally determine the protein structure, like NMR spectroscopy, X–ray crystallography, or cryo–electron tomography, are expensive and overly time consuming. Recently, neural networks have been applied to the secondary structure prediction task, showing promising results. In this scope, LSTMs can focus on both local and global contextual features, and have already been applied to predict secondary structures in proteins with very good results [200, 201].
In this Section, discussing publication [P05], the problem of finding helical moieties in proteins is faced, searching for small conserved amino acid signals that delimitate α–helices. To this aim, we first carried out a statistical analysis, and then implemented and compared three different ML approaches to identify such signals.
Sequences and secondary structure information were extracted using DSSP [172]. In an α–helix, each turn is composed by an average number of 3.6 residues. Therefore, to ensure that each helix includes at least two turns, helices shorter than eight residues were discarded. Since signals that trigger the helix formation can also be located outside the helix sequence itself, we analyzed the sequences, taking into account three amino acids before and after each helix, and four amino acids inside the helix on both sides (as reported in the CATH database [202]), for a total size of 14.
The statistical analysis consisted in measuring the percentage of occurrences of each of the 20 common amino acids in these 14 positions. Known for not being strong helix conformers, Glu, Lys and Arg residues were found to have a low frequency of occurrence inside the helices. Charged amino acids play structural roles in helices, and their frequencies in different positions met the expectations. Pro and Gly have low frequencies in the central positions, whereas they have high concentrations at the ends of the helix. Pro has a

very large side chain, and it is well known that Prolines break α–helix formation. Finally, Ala, Leu, Glu and Met have an especially high propensity to belong to the inner part of helices.
The results of the statistical investigation show that informative patterns can be evidenced at the beginning and at the end of amino acid sequences representing α–helices. In order to validate this assumption, we compared three ML approaches: a random forest classifier, an MLP, and an LSTM. The models rely only on sequence data, and are equipped with attention modules, to decide if the short amino acid sequence of fixed–length previously described can reliably represent an α–helix. Amino acids are encoded either using Word2Vec [203] or one–hot encodings. LSTM is the only model with a true sequential input, while the other two rely on vectorial representations of the sequences (elements side by side). In the random forest model, the attention mechanism is implemented in terms of the importance of each sequence position with respect to the model decision that can be evaluated as an average across the base decision tree classifiers. The MLP model, is equipped with an attention mechanism applied directly on the input, while the LSTM has its attention mechanism attached in two different positions within the network: respectively, after and before the LSTM layer. A permutation between input data axes in the attention mechanism allows us to move the focus on each timestep rather than on each feature. Our primary interest is, in fact, to measure the importance of each sequence position. Secondarily, and only in the one–hot case, the focus on the single feature in each timestep allows us to evaluate the importance of the presence of a particular amino acid in a given position.
After tuning the hyperparameters of each model on a validation set, the models were used in 20 different runs of 10–fold cross–validation in order to measure a reliable average focus of attention. Additionally, a comparison experiment on the test set allowed to determine the model with higher accuracy, which turned out to be the LSTM with the attention mechanism positioned after the LSTM layer. Regardless of the ML model and the encoding used, the attention focuses principally on the last position upstream with respect to the 5' end of the helix, and on the three last positions inside the helix. From the attention weights learned by the networks, it emerges that most of the information which defines the presence of an α–helix is contained in the helix itself. Furthermore, the information at the end of the motif looks more relevant with respect to the previous amino acids. Atten-

tion heatmaps realized on the one–hot encoding input allow to detect the importance of each amino acid in all the 14 positions under analysis. The average attention heatmap obtained in the cross–validation experiment on the LSTM is showed in Fig. 8.2.

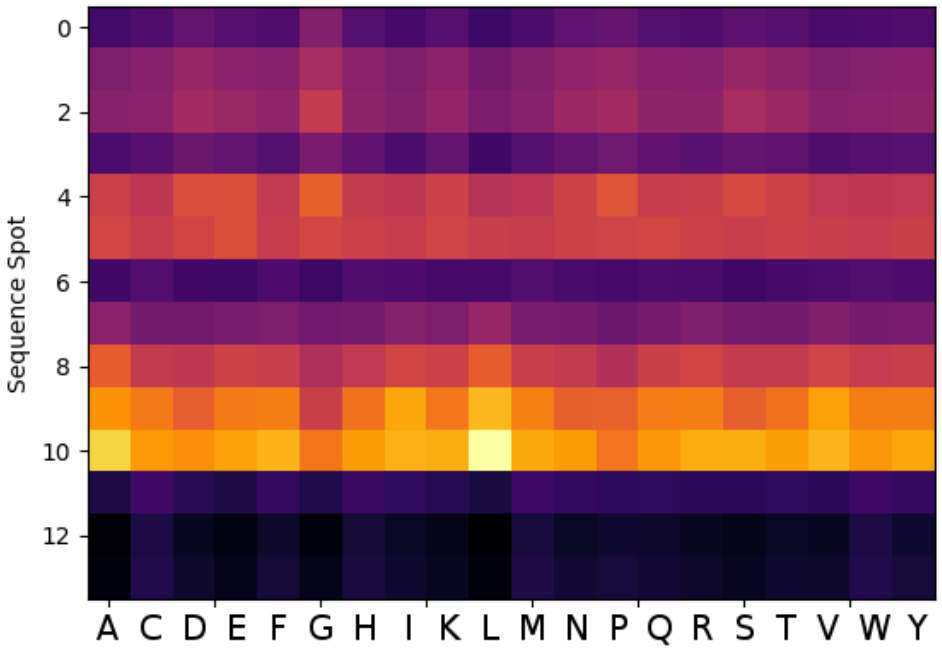


Figure 8.2: Heatmap representing the attention weights learned by the LSTM model with attention before the LSTM layer in the cross–validation experiment. Each row of the heatmap corresponds to a sequence position (from top to bottom), while each column corresponds to one amino acid (indicated by its one letter code).

The focus of attention on the last three positions of the helix is evident. In particular, the LSTM concentrates on the occurrence of the amino acids which are particularly abundant in those positions according to the statistical analysis: Leucine, Alanine, and Valine. In general, the correlation between the heatmap and the results of the previous analysis is evident. The heatmap also underlines the key role of Leucine in α–helix stabilization: The fact that Leucine is the least affected by translation errors, due to its six different codons, seems to make it more preferable than other strong α–helix conformers in the position where helices must collapse. Moreover, it is interesting to notice how the attention is very low on the elements located outside the helix.

The obtained experimental results demonstrate the power of ML techniques in extracting information from protein data to make predictions on the protein structural features, based only on the amino acid sequence. Moreover, having demonstrated that both the statistical and the ML approaches focus on the same positions to ascertain the presence of an α–helix has a twofold impact: on the one hand, it reinforces the biological intuition of the presence of amino acid signals delimiting helical moieties; on the other hand, it ensures the interpretability of the results produced by ML approaches, showing how what is reputed biologically significant is also important for the nework decision.

8.5 Caregiver–Matcher: Graph Neural Networks for Connecting Caregivers of Rare Disease Patients

Any disorder which has a low prevalence in the target population, typically chronic and potentially life–threatening, is known as a rare disease. The estimated number of rare diseases is higher than 6,000 and, depending on the local definitions of rare disease, the prevalence of people suffering from them varies between 3.5% and 5.9%. This results in 263–446 millions of affected people worldwide [204]. Caregivers provide daily assistance to people affected by rare diseases. They can be either members of the patient's family, or people hired for providing help. The constant attention to the patient's needs, and the social isolation that the role of being caregivers entails are at the basis of the obstacles they have to deal with in the daily assistance [205]. In order to cope with the issues of isolation and poor communication with healthcare professionals, a network of caregivers is extremely valuable [206]. In this Section, derived from publication [P14], the proof of concept of a cross–platform application in support of the caregiver's experience is presented. The proposal is called CaregiverMatcher [207], and its aim is to create a network of caregivers of rare disease patients. This project was presented at the Rare Disease Hackathon 2020, organized by the Italian Forum Sistema Salute, reaching the final stage reserved to the best 8 proposals, among more than 250 projects. CaregiverMatcher exploits GNNs [1] to link each caregiver with other caregivers that face similar issues in daily assis-

tance, building a network of contacts, based on information on the assisted patients. Informative sections are integrated into the application, to improve the knowledge about rare diseases.
The core, and main novelty, of CaregiverMatcher is the idea of connecting caregivers with GNNs. From a practical point of view, the application builds links between caregivers based on both patient personal information and health condition. The GNN model is asked to predict whether an edge exists between each pair of caregiver nodes. The predicted presence or absence of an edge represents the existence of a caregiver–caregiver relationship, and it is weighted according to a real–valued similarity score describing how compatible their profiles are: the higher the score, the higher the compatibility between the connected users. Eventually, once the matching process has been completed, the user is returned a list of similar caregivers, filtered as needed by setting some parameters in a dedicated section. This application is thought to be easy to use, and scientific information should be provided in a simple didactic language. CaregiverMatcher has low implementation and maintance costs, compatible with a free application. It is well established that support groups for caregivers lead to improvements in psychological well–being, caregiver burden, and social consequences [208]. Nevertheless, it has to be pointed out that a high level of efficiency of the application can be reached only after a variable (yet not quantifiable) amount of time. The GNN model will require a consistent amount of registered users to learn and efficiently perform a matching between the nodes of the network.
In conclusion, CaregiverMatcher may result in benefits in many aspects of caregivers' life, including mental health, by providing psychological and practical support, along with the possibility to easily access reliable educational material offered by professionals and associations.

8.6 Deep Learning Applications to Image Analysis

This Section presents two DL techniques applied on image data. Sequences of images representing dragonflies are analyzed in order to recognize the actions the dragonflies are performing [209]. Then, a skin lesion classifier based on CNNs is proposed, in order to help the diagnosis of melanoma [210].

8.6.1 Deep Learning Techniques for Dragonfly Action Recognition

This Subsection presents a project, described in publication [P13], for the development of a CNN–based classifier of images of dragonfly flight, capable of recognizing the different phases of the flight. Dragonflies are very complex and advanced flying organisms: their four wings, each of which can be moved independently from the others, give them extraordinary agility. They can fly fast, stand still in mid–air, land and take off in every condition, and they can perform complex acrobatic movements while flying. All these features make dragonfly flight very interesting to study, while also making dragonfly action recognition a difficult task.
More specifically, the proposed model classifies video frames in five categories: take–off, flight, landing, stationary and absent (frames in which the dragonfly is not present). DL requires a huge set of fully annotated data, but, unfortunately, we were not aware of a publicly available labeled dataset of dragonfly images. To train a DL architecture, a suitable number of samples was collected from online videos, which were appropriately pre–processed and labeled frame by frame. Then, two different classifier networks for action recognition were compared: a standard CNN, elaborating one frame at a time, and an LSTM, elaborating sequences of frames.
A first set of experiments was carried out with the aim of identifying a good combination of hyperparameters, such as the number of convolutional blocks or the number of feature maps. Transfer learning was then applied, exploiting pre–trained models that could extract low–level image features efficiently and with short training times. Three CNN models which are well known in the literature were used for transfer learning: the MobileNet–v2 [211], the VGG16 [212] and the DenseNet121 [213]. All of them are pre–trained on the ImageNet dataset [214].
The experimentation on LSTMs was carried out on the same dataset, but taking into account the sequentiality of frames. A CNN–based feature extractor was employed, in order to transform each frame into a feature vector. The sequence of vectors was then analyzed with the LSTM model. The LSTM architecture is characterized by 2 dense layers composed of 100 and 5 neurons, respectively. To analyze the entire sequence, a sliding window with a size of 7 frames was used.
The experimental results showed that both models (CNNs and LSTMs)

reached good accuracy levels, yet with some difficulties in recognizing specific classes. This work demonstrated that DL techniques can be successfully applied to the dragonfly action recognition task. In particular, a good set of guidelines for the automatic analysis of dragonfly flight has been provided. These guidelines include new instructions for setting up a dataset, as well as useful considerations for the calibration, design and implementation of deep models to face this complex task.

8.6.2 Fusion of Visual and Anamnestic Data for the Classification of Skin Lesions with Deep Learning

This Subsection describes the collaboration to a project aimed at classifying skin lesions with CNNs according to their malignancy, developing a tool that could help dermatologists in the diagnosis of melanoma, one of the most common and lethal forms of cancer [215]. The outcome of this project is publication [P12].

More specifically, the study presented in this Section aimed at improving the efficiency in the early detection of skin cancer, developing a classifier capable of integrating information coming from both dermoscopic images and anamnestic data. Experimental tests were carried out on the freely downloadable International Skin Imaging Collaboration (ISIC) archive [216], showing the importance of the exogenous patient data for the correct classification of lesions. The system developed in this work has got a composite architecture that allows to process images and patient data separately and in parallel, with a dedicated module for each one. The partial results are then combined through a third logical unit. The dermoscopic images are analyzed with the *LesionNet*, a deep CNN, while the clinical features are processed by the *MetaNet*, a fully–connected MLP. The outputs of these two networks are concatenated and provided as input to the *MergedNet*, which is trained to classify the lesions by combining the two previous evaluations. In particular, the *LesionNet* is a ResNet [47] with 50 layers, pretrained on ImageNet [214]. The *MetaNet* is a three–layer MLP and processes the vector of clinical features associated with each lesion, composed of: one–hot encoding of the age group, one–hot encoding of lesion location, a patient's gender two–bit flag, and a two–bit code indicating if the lesion is melanocytic or not. The final classification is carried out by the MergedNet. The input of

this network is built by concatenating the output of the last hidden layer of the two specialized networks.
The results of the experimentation showed that, in this task, image–based models perform better than models based only on anamnestic features. This corresponds to the standard medical practice in the diagnosis of melanoma, which is mainly based on the visual inspection of the lesion. The modularity of the model allows to better combine the two approaches, with the anamnestic features helping the decision of the CNN classifier.

Chapter 9

Conclusions and Future Developments

This thesis is focused on GNNs and their applications in molecular data. A software framework for designing and deploying GNNs for research applications has been developed (see Chapter 4). Three applications, relevant both from the point of view of drug discovery and from that of DL, and implemented with our software framework, are presented and discussed. In particular, GNNs are employed for generating molecular graphs (see Chapter 5), predicting the side–effects of drugs (see Chapter 6), and identifying protein–protein interfaces (see Chapter 7). Some specific conclusions on each of these works are drawn at the end of the respective chapters, and summarized in the following.

MG^2N^2, a sequential model for molecular graph generation, is proposed in Chapter 5. Its principal advantage with respect to other sequential models, mainly based on RNNs or RL, is that it exploits a graph representation of the molecule under construction, which is more informative than the sequence of previous actions. The advantage of using a sequential generator, with respect to VAE–based generators, is that sequential models are more interpretable (f.i. errors can be precisely located in the generation steps). The modularity of MG^2N^2 guarantees an easier training procedure, even more precise interpretability (we know which module produced an atom or bond), and the possibility to modify and re–train each module without having to re–train the other ones. Moreover, exploiting the sequential nature of the generative model, it could be possible to take into account the chemical reactions involved in the process, training the model to build only compounds

that can be produced with known chemical reactions. Interpretability has a major role in this scope, as it would become a necessary feature of the model, boosting the usability of sequential models in general, and of modular approaches like MG^2N^2 in particular. Our approach has two main limitations, from which future research directions unfold. On the one hand, MG^2N^2 and similar approaches cannot generate large molecules efficiently: this issue can be addressed with the development of hierarchical generators based on MG^2N^2, that could build larger molecules, such as proteins or polymers. On the other hand, the model generates molecules without specific conditions, which are usually required for the purpose of drug discovery studies. Therefore, an interesting topic of future work is to produce model versions specialized on conditional generation, in which specific features of the compounds to be generated are required in input to the generator.
DruGNN, a GNN–based DSE predictor, is discussed in Chapter 6. Combining data from heterogeneous sources into a relational dataset allows to represent the complex features and interactions involved into the occurrence of DSEs. CGNNs are ideal to process this kind of dataset, following a mixed inductive–transductive learning scheme, and produce accurate predictions in this multi–class multi–label node classification task. Moreover, DruGNN is modular: data can be added without having to re–train the model, and robust to ablation. The proposed method can be easily used to predict the side–effects of a candidate drug, by inserting the molecule and its interaction data. A future research direction consists of specializing the predictor, by taking into account tissue specific data and DSE supervisions, obtaining more accurate and informative insights. Moreover, DruGNN is currently a black–box approach, like many methods based on DNNs. Producing a more interpretable and trustworthy future version of the model would boost the usability of the approach. Another drawback consists of the approximation of structural information introduced by fingerprints. On the one hand, structural embeddings could be extracted with a GNN from the structural formulas of drug molecules. On the other hand, it is a very interesting matter of future work to develop a similar predictor that takes in input the structural formulas of molecules, exploiting the full chemical graph information. Features and interactions can then be added in the same shape used for DruGNN, exploiting an approach based on a graph (the relational dataset) of graphs (the structural formulas of drugs).
GNNs were also employed for predicting protein–protein interfaces, as de-

scribed in Chapter 7. This task was formulated as a maximum clique search on the correspondence graph derived from the secondary structures of each pair of proteins. The approach showed very promising results, individuating the secondary structures that participate in each interface, and providing interesting insights for future research. A first limitation is that current results are preliminary: an improved version of the model could be developed based on a more extensive experimentation, also taking into account more possible GNN models and architectures. The second drawback is represented by the coarse scale of the interface representation predicted. In fact, the same analysis could be refined to the amino acid scale, or even at the atom level, more precisely identifying the residues that take part into protein–protein interfaces.

These three applications constitute demonstrations of the capabilities of GNNs for molecular data in different settings, relevant to the field of drug discovery, such as graph generation, node classification in a classical yet imbalanced setting, and node classification on a complex, heterogeneous relational dataset. Moreover, the properties of recurrent GNNs were made easily available for research through the development of a software framework that allows an easy Keras–based implementation. As a general conclusion, the demonstrations obtained of GNN properties on these testing grounds, and the general growth of the scientific interest and community related to graph–based models, suggest that GNNs will continue to improve. New theories and models will certainly be proposed, also concerning the possibility of non–local GNNs, capable of reaching higher computational power (f.i. simulating higher order Weisfeiler–Lehman tests). New application fields, and even more complex data settings, will be addressed with GNNs in the future, providing fundamental contributions to many scientific fields, ranging a wide variety of disciplines, such as biology, physics simulations, weather prediction, and social network analysis.

Publications and Activities

Journal Articles

P01 **Pietro Bongini**, Monica Bianchini, Franco Scarselli. (2021). Molecular generative Graph Neural Networks for Drug Discovery. Neurocomputing, 450, 242–252.
https://doi.org/10.1016/j.neucom.2021.04.039
Candidate's contributions: original idea (in collaboration), model and algorithm design, software implementation, experimentation, original manuscript draft, manuscript reviewing and editing.
Thesis relevance: primary contribution, presented in Chapter 5.

P02 **Pietro Bongini**, Franco Scarselli, Monica Bianchini, Giovanna Maria Dimitri, Niccolò Pancino, Pietro Liò. (2022). Modular multi–source prediction of drug side–effects with DruGNN. Published in early access on IEEE/ACM Transactions on Computational Biology and Bioinformatics.
https://doi.org/10.1109/TCBB.2022.3175362
Candidate's contributions: conceptualization, software implementation, dataset construction, experimentation, original manuscript draft, manuscript reviewing and editing.
Thesis relevance: primary contribution, presented in Chapter 6.

P03 Niccolò Pancino, **Pietro Bongini**, Franco Scarselli, Monica Bianchini. (2022). GNNkeras: A Keras–based library for Graph Neural Networks and homogeneous and heterogeneous graph processing. SoftwareX, 18, 101061.
https://doi.org/10.1016/j.softx.2022.101061
Candidate's contributions: software implementation (in collaboration), manuscript reviewing and editing.

Thesis relevance: software contribution, presented in Chapter 4.

P04 **Pietro Bongini**, Alfonso Trezza, Monica Bianchini, Ottavia Spiga, Neri Niccolai. (2020). A possible strategy to fight COVID-19: interfering with spike glycoprotein trimerization. Biochemical and biophysical research communications, 528(1), 35–38.
https://doi.org/10.1016/j.bbrc.2020.04.007
Candidate's contributions: algorithm design, software implementation, experimentation (protein structural analysis, transient pocket detection and evaluation), manuscript reviewing and editing.
Thesis relevance: other works, discussed in Subsection 8.3.1.

P05 Anna Visibelli, **Pietro Bongini**, Alberto Rossi, Neri Niccolai, Monica Bianchini. (2020). A deep attention network for predicting amino acid signals in the formation of α-helices. Journal of Bioinformatics and Computational Biology, 18(5), 2050028.
https://doi.org/10.1142/S0219720020500286
Candidate's contributions: model design (attention mechanisms), experimentation (in collaboration), data visualization (attention graphs and heatmaps), manuscript reviewing and editing.
Thesis relevance: other works, discussed in Section 8.4.

P06 Luca Oneto, Nicolò Navarin, Battista Biggio, Federico Errica, Alessio Micheli, Franco Scarselli, Monica Bianchini, Luca Demetrio, **Pietro Bongini**, Armando Tacchella, Alessandro Sperduti. (2022). Towards Learning Trustworthily, Automatically, and with Guarantees on Graphs: an Overview. Neurocomputing, 493, 217–243.
https://doi.org/10.1016/j.neucom.2022.04.072
Candidate's contributions: manuscript writing (section on the computational capabilities of Graph Neural Networks), literature survey and integration (in collaboration), manuscript reviewing and editing.
Thesis relevance: other works, discussed in Section 8.1.

P07 **Pietro Bongini**, Neri Niccolai, Monica Bianchini. (2019). Glycine-induced formation and druggability score prediction of protein surface pockets. Journal of Bioinformatics and Computational Biology, 17(5), 1950026.
https://doi.org/10.1142/S0219720019500264
Candidate's contributions: algorithm and model design, software

implementation, experimentation, original manuscript draft, manuscript reviewing and editing.
Thesis relevance: other works, discussed in Subsection 8.2.1.

P08 **Pietro Bongini**, Simone Gardini, Monica Bianchini, Ottavia Spiga, Neri Niccolai. (2021). Structural bioinformatics survey on disease-inducing missense mutations. Journal of Bioinformatics and Computational Biology, 19(3), 2150008.
https://doi.org/10.1142/S0219720021500086
Candidate's contributions: algorithm design, data analysis, software implementation, experimentation, manuscript reviewing and editing.
Thesis relevance: other works, discussed in Subsection 8.2.2.

P09 **Pietro Bongini**, Neri Niccolai, Alfonso Trezza, Guido Mangiavacchi, Annalisa Santucci, Ottavia Spiga, Monica Bianchini, Simone Gardini. (2020). Structural bioinformatic survey of protein-small molecule interfaces delineates the role of glycine in surface pocket formation. IEEE/ACM Transactions on Computational Biology and Bioinformatics.
https://doi.org/10.1109/TCBB.2020.3033384
Candidate's contributions: data analysis, software implementation, experimentation (in collaboration), manuscript reviewing and editing.
Thesis relevance: other works, discussed in Subsection 8.2.3.

P10 Vittoria Cicaloni, Filippo Costanti, Arianna Pasqui, Monica Bianchini, Neri Niccolai, **Pietro Bongini**. (2022). A bioinformatic approach to investigate structural and non–structural proteins in human coronaviruses. Accepted for publication on Frontiers in Genetics.
https://doi.org/10.3389/fgene.2022.891418
Candidate's contributions: supervision, software implementation (in collaboration), original manuscript draft (computational part), manuscript reviewing and editing.
Thesis relevance: other works, discussed in Subsection 8.3.2.

Conference Papers

P11 Niccolò Pancino, Alberto Rossi, Giorgio Ciano, Giorgia Giacomini, Simone Bonechi, Paolo Andreini, Franco Scarselli, Monica Bianchini, **Pietro Bongini**. (2020). Graph Neural Networks for the Prediction of Protein-Protein Interfaces. In: 28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2020), 127–132.
Candidate's contributions: supervision, model and algorithm design (in collaboration with Niccolò Pancino), software implementation (in collaboration with Niccolò Pancino), manuscript reviewing and editing.
Thesis relevance: primary contribution, presented in Chapter 7.

P12 Simone Bonechi, Monica Bianchini, **Pietro Bongini**, Giorgio Ciano, Giorgia Giacomini, Riccardo Rosai, Linda Tognetti, Alberto Rossi, Paolo Andreini. (2019). Fusion of visual and anamnestic data for the classification of skin lesions with deep learning. In: 20th International Conference on Image Analysis and Processing (ICIAP 2019), 211–219.
Candidate's contributions: conceptualization and discussion (in collaboration), data analysis (in collaboration), manuscript reviewing and editing.
Thesis relevance: other works, discussed in Subsection 8.6.2.

P13 Martina Monaci, Niccolò Pancino, Paolo Andreini, Simone Bonechi, **Pietro Bongini**, Alberto Rossi, Giorgio Ciano, Giorgia Giacomini, Franco Scarselli, Monica Bianchini. (2020). Deep Learning Techniques for Dragonfly Action Recognition. In: Proceedings of the 9th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2020), 1, 562–569.
Candidate's contributions: conceptualization and discussion (in collaboration), data analysis (in collaboration), manuscript reviewing and editing.
Thesis relevance: other works, discussed in Subsection 8.6.1.

P14 Filippo Guerranti, Mirco Mannino, Federica Baccini, **Pietro Bongini**, Niccolò Pancino, Anna Visibelli, Sara Marziali. (2021). CaregiverMatcher: graph neural networks for connecting caregivers of rare disease patients. In: 25th International Conference on Knowledge-Based

and Intelligent Information & Engineering Systems (KES 2021). Procedia Computer Science, 192, 1696–1704.
https://doi.org/10.1016/j.procs.2021.08.174
Candidate's contributions: model and algorithm design (in collaboration), conceptualization and discussion (in collaboration), original manuscript draft (sections on Graph Neural Networks and their application), manuscript reviewing and editing, conference presentation.
Thesis relevance: other works, discussed in Section 8.5.

Book Chapters

P15 **Pietro Bongini**, Alfonso Trezza, Monica Bianchini, Ottavia Spiga, Neri Niccolai. (2021). Structural Bioinformatics to Unveil Weaknesses of Coronavirus Spike Glycoprotein Stability. In: Roy K. In Silico Modeling of Drugs Against Coronaviruses. Methods in Pharmacology and Toxicology. Humana, New York, NY.
https://doi.org/10.1007/7653_2020_59
Candidate's contributions: original work, book chapter reviewing and editing.
Thesis relevance: other works, discussed in Subsection 8.3.1.

P16 **Pietro Bongini**, Niccolò Pancino, Franco Scarselli, Monica Bianchini. (2022). BioGNN: How Graph Neural Networks can solve biological problems. In: Cheng Peng Lim et al. Handbook of Artificial Intelligence in Healthcare. Intelligent Systems Reference Library. Springer Nature Switzerland, Cham, CH.

Acknowledgements

First of all, I would like to express all my gratitude to my advisors Prof. Monica Bianchini and Prof. Franco Scarselli: thank you very much for believing in me and for supporting me through all my projects. They showed me how beautiful research is, and taught me how to make it. I am very grateful to them for sharing their precious ideas and expertise, and for their countless helpful suggestions.
I would like to deeply thank Prof. Pietro Liò, for the very stimulating collaboration we had. His ideas are always a spur for finding new ways of making research and improving the existing methods.
My gratitude goes to Prof. Neri Niccolai. He has been a great guide to the field of bioinformatics. Many of my projects would have never seen the light without his deep knowledge of structural biology.
I would like to thank Prof. Fabio Aiolli and Prof. Andrea Passerini for being part of my supervisory committee, and for all their suggestions and opinions.
A special thank you goes to all my fellow Ph.D. students at the Siena Artificial Intelligence Laboratory (SAILab): sharing these three years with them was stimulating from the academic point of view, and a pleasurable experience all–round.
A particular mention goes to my friend and colleague Niccolò Pancino: it was great to work together in many projects, a good part of which would have not been possible without his ideas and his programming skills.
I want to thank the institutions who made these studies possible: the Ph.D program in Smart Computing, co-financed by the Universities of Florence, Siena, and Pisa, and by Regione Toscana under the European Social Fund; the University of Florence, which hosts the Ph.D. program; the University of Siena, and in particular the Department of Information Engineering and Mathematics, where I carried out most of my research work.
My deepest gratitude goes to my family, for believing in me, supporting me

through every path I followed, and for helping me to shape my life and career. I want to say a warm and hearty thank you to my mother Angela and my father Antonio, to my brother Marco and to Erjona and to my nephew Fabio, to my grandmother Leda, and to all the rest of the family.
My most special gratitude goes to Stella: her support, empathy, and love are fundamental to me. Thank you with all my heart for believing in me, for understanding me, and for encouraging me. I am very lucky, grateful, and proud to have you.
Finally, I would like to thank all the friends and relatives who supported me through my academic career and all the other aspects of my life.

Bibliography

[1] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[2] M. Wang and G. Hu, "A novel method for twitter sentiment analysis based on attentional–graph neural network," *Information*, vol. 11, no. 2, p. 92, 2020.

[3] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Hierarchical representation learning in graph neural networks with node decimation pooling," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[4] N. Pancino, A. Rossi, G. Ciano, G. Giacomini, S. Bonechi, P. Andreini, F. Scarselli, M. Bianchini, and P. Bongini, "Graph neural networks for the prediction of protein–protein interfaces," in *ESANN*, pp. 127–132, 2020.

[5] B. Donon, B. Donnot, I. Guyon, and A. Marot, "Graph neural solver for power systems," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2019.

[6] N. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[7] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "Computational capabilities of graph neural networks," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 81–102, 2009.

[8] F. Scarselli, S. L. Yong, M. Gori, M. Hagenbuchner, A. C. Tsoi, and M. Maggini, "Graph neural networks for ranking web pages," in *The*

2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05), pp. 666–672, IEEE, 2005.

[9] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[10] J. Kim, S. Park, D. Min, and W. Kim, "Comprehensive survey of recent drug discovery using deep learning," *International Journal of Molecular Sciences*, vol. 22, no. 18, p. 9983, 2021.

[11] P. Bongini, M. Bianchini, and F. Scarselli, "Molecular generative graph neural networks for drug discovery," *Neurocomputing*, vol. 450, pp. 242–252, 2021.

[12] E. Dai and S. Wang, "Towards self–explainable graph neural network," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 302–311, 2021.

[13] Z. Zhong, C.-T. Li, and J. Pang, "Hierarchical message-passing graph neural networks," 2020.

[14] N. Pancino, P. Bongini, F. Scarselli, and M. Bianchini, "Gnnkeras: A keras-based library for graph neural networks and homogeneous and heterogeneous graph processing," *SoftwareX*, vol. 18, p. 101061, 2022.

[15] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[18] C. Cortes and V. Vapnik, "Support–vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[19] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[20] T. Kohonen, "Self–organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[21] J. A. Davis, "Clustering and structural balance in graphs," *Human relations*, vol. 20, no. 2, pp. 181–187, 1967.

[22] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1, pp. 278–282, IEEE, 1995.

[23] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[24] Y. Bengio, P. Simard, and P. Frasconi, "Learning long–term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[25] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[26] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[27] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[28] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological cybernetics*, vol. 20, no. 3, pp. 121–136, 1975.

[29] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[30] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[31] S. Hochreiter and J. Schmidhuber, "Long short–term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[32] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[33] K. M. Tarwani and S. Edem, "Survey on recurrent neural network in natural language processing," *Int. J. Eng. Trends Technol*, vol. 48, pp. 301–304, 2017.

[34] S. K. Sønderby and O. Winther, "Protein secondary structure prediction with long short term memory networks," 2014.

[35] A. Visibelli, P. Bongini, A. Rossi, N. Niccolai, and M. Bianchini, "A deep attention network for predicting amino acid signals in the formation of α-helices," *Journal of Bioinformatics and Computational Biology*, vol. 18, no. 05, p. 2050028, 2020.

[36] K. Pawar, R. S. Jalem, and V. Tiwari, "Stock market price prediction using lstm rnn," in *Emerging Trends in Expert Applications and Security*, pp. 493–503, Springer, 2019.

[37] M. Weber, M. Liwicki, D. Stricker, C. Scholzel, and S. Uchida, "Lstm–based early recognition of motion patterns," in *2014 22nd International Conference on Pattern Recognition*, pp. 3552–3557, IEEE, 2014.

[38] A. Graves, N. Jaitly, and A. rahman Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *2013 IEEE workshop on automatic speech recognition and understanding*, pp. 273–278, IEEE, 2013.

[39] J. Zhao, X. Mao, and L. Chen, "Speech emotion recognition using deep 1d & 2d cnn lstm networks," *Biomedical Signal Processing and Control*, vol. 47, pp. 312–323, 2019.

[40] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[42] Y. LeCun *et al.*, "Generalization and network design strategies," *Connectionism in perspective*, vol. 19, pp. 143–155, 1989.

[43] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5455–5516, 2020.

[44] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[45] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[46] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[48] D. Aldous and J. Fill, "Reversible markov chains and random walks on graphs," 2002.

[49] D. Haussler, "Convolution kernels on discrete structures," tech. rep., Technical report, Department of Computer Science, University of California, 1999.

[50] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi, "Graph kernels for chemical informatics," *Neural networks*, vol. 18, no. 8, pp. 1093–1110, 2005.

[51] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 714–735, 1997.

[52] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE transactions on Neural Networks*, vol. 9, no. 5, pp. 768–786, 1998.

[53] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, pp. 1024–1034, 2017.

[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[55] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, "Why and when can deep–but not shallow–networks avoid the curse of dimensionality: a review," *International Journal of Automation and Computing*, vol. 14, no. 5, pp. 503–519, 2017.

[56] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *International Conference on Learning Representations*, 2018.

[57] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI Series*, vol. 2, no. 9, pp. 12–16, 1968.

[58] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4602–4609, 2019.

[59] G. A. D'Inverno, M. Bianchini, M. L. Sampoli, and F. Scarselli, "An unifying point of view on expressive power of gnns," 2021.

[60] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.

[61] N. Bandinelli, M. Bianchini, and F. Scarselli, "Learning long–term dependencies using layered graph neural networks," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2010.

[62] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," 2018.

[63] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," 2015.

[64] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 1263–1272, 2017.

[65] T. N. Kipf and M. Welling, "Semi–supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017*, 2017.

[66] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

[67] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, pp. 3844–3852, 2016.

[68] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017.

[69] S. Munikoti, L. Das, and B. Natarajan, "Scalable graph neural network-based framework for identifying critical nodes and links in complex networks," *Neurocomputing*, vol. 468, pp. 211–221, 2022.

[70] F. Scarselli, A. C. Tsoi, M. Hagenbuchner, and L. Di Noi, "Solving graph data issues using a layered architecture approach with applications to web spam detection," *Neural Networks*, vol. 48, pp. 78–90, 2013.

[71] Z. Li, Z. Cui, S. Wu, X. Zhang, and L. Wang, "Fi-gnn: Modeling feature interactions via graph neural networks for ctr prediction," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 539–548, 2019.

[72] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," 2020.

[73] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 165–174, 2019.

[74] X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua, "Kgat: Knowledge graph attention network for recommendation," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 950–958, 2019.

[75] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The World Wide Web Conference*, pp. 2022–2032, 2019.

[76] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *International conference on machine learning*, pp. 3835–3845, PMLR, 2019.

[77] A. M. Karimi, Y. Wu, M. Koyuturk, and R. H. French, "Spatiotemporal graph neural network for performance prediction of photovoltaic power systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 15323–15330, 2021.

[78] G. Bécigneul, O.-E. Ganea, B. Chen, R. Barzilay, and T. Jaakkola, "Optimal transport graph neural networks," 2020.

[79] B. Wu, Y. Liu, B. Lang, and L. Huang, "Dgcnn: Disordered graph convolutional neural network based on the gaussian mixture model," *Neurocomputing*, vol. 321, pp. 346–356, 2018.

[80] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1666–1674, 2018.

[81] O. J. Wouters, M. McKee, and J. Luyten, "Estimated research and development investment needed to bring a new medicine to market, 2009-2018," *Jama*, vol. 323, no. 9, pp. 844–853, 2020.

[82] M. L. Billingsley, "Druggable targets and targeted drugs: Enhancing the development of new therapeutics," *Pharmacology*, vol. 82, no. 4, pp. 239–244, 2008.

[83] M. Dickson and J. P. Gagnon, "Key factors in the rising cost of new drug discovery and development," *Nature reviews Drug discovery*, vol. 3, no. 5, pp. 417–429, 2004.

[84] S. J. Y. Macalino, V. Gosu, S. Hong, and S. Choi, "Role of computer-aided drug design in modern drug discovery," *Archives of pharmacal research*, vol. 38, no. 9, pp. 1686–1701, 2015.

[85] J. S. Smith, A. E. Roitberg, and O. Isayev, "Transforming computational drug discovery with machine learning and ai," *ACS medicinal chemistry letters*, vol. 9, no. 11, pp. 1065–1069, 2018.

[86] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[87] C. F. Lipinski, V. G. Maltarollo, P. R. Oliveira, A. B. da Silva, and K. M. Honorio, "Advances and perspectives in applying deep learning for drug design and discovery," *Frontiers in Robotics and AI*, vol. 6, p. 108, 2019.

[88] A. Zhavoronkov, Y. A. Ivanenkov, A. Aliper, M. S. Veselov, V. A. Aladinskiy, A. V. Aladinskaya, V. A. Terentiev, D. A. Polykovskiy,

M. D. Kuznetsov, A. Asadulaev, *et al.*, "Deep learning enables rapid identification of potent ddr1 kinase inhibitors," *Nature biotechnology*, vol. 37, no. 9, pp. 1038–1040, 2019.

[89] Q. Feng, E. Dueva, A. Cherkasov, and M. Ester, "Padme: A deep learning-based framework for drug-target interaction prediction," 2018.

[90] T. B. Kimber, Y. Chen, and A. Volkamer, "Deep learning in virtual screening: Recent applications and developments," *International Journal of Molecular Sciences*, vol. 22, no. 9, p. 4435, 2021.

[91] M. Skalic, A. Varela-Rial, J. Jiménez, G. Martínez-Rosell, and G. De Fabritiis, "Ligvoxel: inpainting binding pockets using 3d-convolutional neural networks," *Bioinformatics*, vol. 35, no. 2, pp. 243–250, 2019.

[92] P. Bongini, N. Niccolai, and M. Bianchini, "Glycine–induced formation and druggability score prediction of protein surface pockets," *Journal of bioinformatics and computational biology*, vol. 17, no. 05, p. 1950026, 2019.

[93] X. Zeng, S. Zhu, W. Lu, Z. Liu, J. Huang, Y. Zhou, J. Fang, Y. Huang, H. Guo, L. Li, *et al.*, "Target identification among known drugs by deep learning from heterogeneous networks," *Chemical Science*, vol. 11, no. 7, pp. 1775–1797, 2020.

[94] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, vol. 4, no. 2, pp. 268–276, 2018.

[95] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *International conference on machine learning*, pp. 2323–2332, PMLR, 2018.

[96] P. ErdḦos and A. Rényi, "On random graphs I," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959.

[97] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Mod. Phys.*, vol. 74, pp. 47–97, 2002.

[98] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small–world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[99] D. P. Kingma and M. Welling, "Auto–encoding variational bayes," 2013.

[100] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *NeurIPS Bayesian Deep Learning Workshop*, 2016.

[101] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *International Conference on Machine Learning*, pp. 2434–2444, 2019.

[102] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "GraphGAN: Graph representation learning with generative adversarial nets," in *Thirty-second AAAI conference on artificial intelligence*, pp. 2508–2515, 2018.

[103] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating graphs via random walks," in *International Conference on Machine Learning*, pp. 610–619, 2018.

[104] L. Di Liello, P. Ardino, J. Gobbi, P. Morettin, S. Teso, and A. Passerini, "Efficient generation of structured objects with constrained adversarial networks," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 14663–14674, Curran Associates, Inc., 2020.

[105] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto–regressive models," in *International Conference on Machine Learning*, pp. 5708–5717, 2018.

[106] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel, "Efficient graph generation with graph recurrent attention networks," in *Advances in Neural Information Processing Systems*, pp. 4255–4265, 2019.

[107] D. Bacciu, A. Micheli, and M. Podda, "Edge–based sequential graph generation with recurrent neural networks," *Neurocomputing*, vol. 416, pp. 177–189, 2020.

[108] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," 2018.

[109] D. Weininger, "Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules," *Journal of chemical information and computer sciences*, vol. 28, no. 1, pp. 31–36, 1988.

[110] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *International Conference on Machine Learning*, pp. 1945–1954, PMLR, 2017.

[111] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song, "Syntax–directed variational autoencoder for structured data," *arXiv preprint arXiv:1802.08786*, 2018.

[112] X. Yang, J. Zhang, K. Yoshizoe, K. Terayama, and K. Tsuda, "Chemts: an efficient python library for de novo molecular generation," *Science and technology of advanced materials*, vol. 18, no. 1, pp. 972–976, 2017.

[113] M. Simonovsky and N. Komodakis, "GraphVAE: Towards generation of small graphs using variational autoencoders," in *International Conference on Artificial Neural Networks*, pp. 412–422, Springer, 2018.

[114] W. Jin, R. Barzilay, and T. Jaakkola, "Hierarchical generation of molecular graphs using structural motifs," in *International Conference on Machine Learning*, pp. 4839–4848, PMLR, 2020.

[115] D. Rigoni, N. Navarin, and A. Sperduti, "Conditional constrained graph variational autoencoders for molecule design," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 729–736, IEEE, 2020.

[116] C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang, "GraphAF: a flow–based autoregressive model for molecular graph generation," in *7th International Conference on Learning Representations (ICLR)*, 2019.

[117] N. De Cao and T. Kipf, "MolGAN: An implicit generative model for small molecular graphs," 2018.

[118] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal–directed molecular graph generation," in *Advances in neural information processing systems*, pp. 6410–6421, 2018.

[119] Y. Yamanishi, M. Kotera, M. Kanehisa, and S. Goto, "Drug–target interaction prediction from chemical, genomic and pharmacological data in an integrated framework," *Bioinformatics*, vol. 26, no. 12, pp. i246–i254, 2010.

[120] S. Mizutani, E. Pauwels, V. Stoven, S. Goto, and Y. Yamanishi, "Relating drug–protein interaction network with drug side effects," *Bioinformatics*, vol. 28, no. 18, pp. i522–i528, 2012.

[121] W. Zhang, Y. Chen, S. Tu, F. Liu, and Q. Qu, "Drug side effect prediction through linear neighborhoods and multiple data source integration," in *2016 IEEE international conference on bioinformatics and biomedicine (BIBM)*, pp. 427–434, IEEE, 2016.

[122] I. Shaked, M. A. Oberhardt, N. Atias, R. Sharan, and E. Ruppin, "Metabolic network prediction of drug side effects," *Cell systems*, vol. 2, no. 3, pp. 209–213, 2016.

[123] G. M. Dimitri and P. Liò, "DrugClust: A machine learning approach for drugs side effects prediction," *Computational biology and chemistry*, vol. 68, pp. 204–210, 2017.

[124] E. Pauwels, V. Stoven, and Y. Yamanishi, "Predicting drug side–effect profiles: A chemical fragment–based approach," *BMC bioinformatics*, vol. 12, no. 1, pp. 1–13, 2011.

[125] A. Cakir, M. Tuncer, H. Taymaz-Nikerel, and O. Ulucan, "Side effect prediction based on drug–induced gene expression profiles and random forest with iterative feature selection," *The Pharmacogenomics Journal*, pp. 1–9, 2021.

[126] O. C. Uner, R. G. Cinbis, O. Tastan, and A. E. Cicek, "DeepSide: A deep learning framework for drug side effect prediction," 2019.

[127] A. Deac, Y.-H. Huang, P. Veličković, P. Liò, and J. Tang, "Drug–drug adverse effect prediction with graph co–attention," 2019.

[128] M. Zitnik, M. Agrawal, and J. Leskovec, "Modeling polypharmacy side effects with graph convolutional networks," *Bioinformatics*, vol. 34, no. 13, pp. i457–i466, 2018.

[129] J. Janin, R. P. Bahadur, and P. Chakrabarti, "Protein–protein interaction and quaternary structure," *Quarterly reviews of biophysics*, vol. 41, no. 2, pp. 133–180, 2008.

[130] T. J. Lane, D. Shukla, K. A. Beauchamp, and V. S. Pande, "To milliseconds and beyond: challenges in the simulation of protein folding," *Current opinion in structural biology*, vol. 23, no. 1, pp. 58–65, 2013.

[131] L. C. Xue, D. Dobbs, and V. Honavar, "Homppi: a class of sequence homology based protein-protein interface prediction methods," *BMC bioinformatics*, vol. 12, no. 1, pp. 1–24, 2011.

[132] H. Hwang, D. Petrey, and B. Honig, "A hybrid method for protein–protein interface prediction," *Protein Science*, vol. 25, no. 1, pp. 159–165, 2016.

[133] H. Hwang, T. Vreven, and Z. Weng, "Binding interface prediction by combining protein–protein docking results," *Proteins: Structure, Function, and Bioinformatics*, vol. 82, no. 1, pp. 57–66, 2014.

[134] J. R. Bradford and D. R. Westhead, "Improved prediction of protein–protein binding sites using a support vector machines approach," *Bioinformatics*, vol. 21, no. 8, pp. 1487–1494, 2005.

[135] K. Huang, C. Xiao, L. M. Glass, M. Zitnik, and J. Sun, "Skipgnn: predicting molecular interactions with skip-graph networks," *Scientific reports*, vol. 10, no. 1, pp. 1–16, 2020.

[136] Y. Liu, H. Yuan, L. Cai, and S. Ji, "Deep learning of high–order interactions for protein interface prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 679–687, 2020.

[137] E. J. Gardiner, P. J. Artymiuk, and P. Willett, "Clique–detection algorithms for matching three-dimensional molecular structures," *Journal of Molecular Graphics and Modelling*, vol. 15, no. 4, pp. 245–253, 1997.

[138] H. M. Grindley, P. J. Artymiuk, D. W. Rice, and P. Willett, "Identification of tertiary structure resemblance in proteins using a maximal common subgraph isomorphism algorithm," *Journal of molecular biology*, vol. 229, no. 3, pp. 707–721, 1993.

[139] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, pp. 729–734, IEEE, 2005.

[140] A. Rossi, M. Tiezzi, G. M. Dimitri, M. Bianchini, M. Maggini, and F. Scarselli, "Inductive–transductive learning with graph neural networks," in *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pp. 201–212, Springer, 2018.

[141] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.

[142] G. Ciano, A. Rossi, M. Bianchini, and F. Scarselli, "On inductive–transductive learning with graph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[143] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel–softmax," in *5th International Conference on Learning Representations (ICLR)*, 2017.

[144] E. J. Gumbel, "Statistical theory of extreme values and some practical applications: a series of lectures," 1954.

[145] C. J. Maddison, D. Tarlow, and T. Minka, "A* sampling," in *Advances in Neural Information Processing Systems*, pp. 3086–3094, 2014.

[146] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977.

[147] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Scientific Data*, vol. 1, 2014.

[148] J. J. Irwin and B. K. Shoichet, "Zinc – a free database of commercially available compounds for virtual screening," *Journal of chemical information and modeling*, vol. 45, no. 1, pp. 177–182, 2005.

[149] L. Ruddigkeit, R. van Deursen, L. C. Blum, and J.-L. Reymond, "Enumeration of 166 billion organic small molecules in the chemical universe database GDB–17," *J. Chem. Inf. Model.*, vol. 52, no. 11, pp. 2864–2875, 2012.

[150] S. A. Wildman and G. M. Crippen, "Prediction of physicochemical parameters by atomic contributions," *Journal of Chemical Information and Computer Sciences*, vol. 39, no. 5, pp. 868–873, 1999.

[151] G. R. Bickerton, G. V. Paolini, J. Besnard, S. Muresan, and A. L. Hopkins, "Quantifying the chemical beauty of drugs," *Nature chemistry*, vol. 4, no. 2, pp. 90–98, 2012.

[152] A. Hagberg, P. Swart, and D. S. Chult, "Exploring network structure, dynamics, and function using networkX," in *In Proceedings of the 7th Python in Science Conference (SciPy)* (J. M. G. Varoquaux, T. Vaught, ed.), pp. 11–15, 2008.

[153] D. Flam-Shepherd, T. Wu, and A. Aspuru-Guzik, "Graph deconvolutional generation," 2020.

[154] Q. Liu, M. Allamanis, M. Brockschmidt, and A. Gaunt, "Constrained graph variational autoencoders for molecule design," *Advances in Neural Information Processing Systems*, vol. 31, pp. 7795–7804, 2018.

[155] H. Khalil and C. Huang, "Adverse drug reactions in primary care: A scoping review," *BMC health services research*, vol. 20, no. 1, pp. 1–13, 2020.

[156] F. R. Ernst and A. J. Grizzle, "Drug–related morbidity and mortality: Updating the cost–of–illness model," *Journal of the American Pharmaceutical Association*, vol. 41, no. 2, pp. 192–199, 2001.

[157] E. D. Kantor, C. D. Rehm, J. S. Haas, A. T. Chan, and E. L. Giovannucci, "Trends in prescription drug use among adults in the United States from 1999–2012," *Jama*, vol. 314, no. 17, pp. 1818–1830, 2015.

[158] M. Kuhn, I. Letunic, L. J. Jensen, and P. Bork, "The SIDER database of drugs and side effects," *Nucleic acids research*, vol. 44, no. D1, pp. D1075–D1079, 2016.

[159] K. Luck, D.-K. Kim, L. Lambourne, K. Spirohn, B. E. Begg, W. Bian, R. Brignall, T. Cafarelli, F. J. Campos-Laborie, B. Charloteaux, *et al.*, "A reference map of the human binary protein interactome," *Nature*, vol. 580, no. 7803, pp. 402–408, 2020.

[160] D. Smedley, S. Haider, S. Durinck, L. Pandini, P. Provero, J. Allen, O. Arnaiz, M. H. Awedh, R. Baldock, G. Barbiera, *et al.*, "The BioMart community portal: An innovative alternative to large, centralized data repositories," *Nucleic acids research*, vol. 43, no. W1, pp. W589–W598, 2015.

[161] D. Szklarczyk, A. Santos, C. Von Mering, L. J. Jensen, P. Bork, and M. Kuhn, "STITCH 5: Augmenting protein–chemical interaction networks with tissue and affinity data," *Nucleic acids research*, vol. 44, no. D1, pp. D380–D384, 2016.

[162] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, *et al.*, "PubChem in 2021: New data content and improved web interfaces," *Nucleic acids research*, vol. 49, no. D1, pp. D1388–D1395, 2021.

[163] T. T. Tanimoto, "IBM internal report 17th," tech. rep., IBM, 11 1957.

[164] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, *et al.*, "Gene ontology: Tool for the unification of biology," *Nature genetics*, vol. 25, no. 1, pp. 25–29, 2000.

[165] D. W. Huang, B. T. Sherman, and R. A. Lempicki, "Systematic and integrative analysis of large gene lists using DAVID bioinformatics resources," *Nature protocols*, vol. 4, no. 1, pp. 44–57, 2009.

[166] D. W. Huang, B. T. Sherman, and R. A. Lempicki, "Bioinformatics enrichment tools: Paths toward the comprehensive functional analysis of large gene lists," *Nucleic acids research*, vol. 37, no. 1, pp. 1–13, 2009.

[167] H. Hegyi and M. Gerstein, "The relationship between protein structure and function: a comprehensive survey with application to the yeast genome," *Journal of molecular biology*, vol. 288, no. 1, pp. 147–164, 1999.

[168] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, "The maximum clique problem," in *Handbook of combinatorial optimization*, pp. 1–74, Springer, 1999.

[169] E. Krissinel, "Crystal contacts as nature's docking solutions," *Journal of computational chemistry*, vol. 31, no. 1, pp. 133–143, 2010.

[170] T. Schäfer, P. May, and I. Koch, "Computation and visualization of protein topology graphs including ligand information," in *German Conference on Bioinformatics 2012*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[171] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The protein data bank," *Nucleic acids research*, vol. 28, no. 1, pp. 235–242, 2000.

[172] W. Kabsch and C. Sander, "Dictionary of protein secondary structure: Pattern recognition of hydrogen–bonded and geometrical features," *Biopolymers*, vol. 22, no. 12, pp. 2577–2637, 1983.

[173] J. Kyte and R. F. Doolittle, "A simple method for displaying the hydropathic character of a protein," *Journal of molecular biology*, vol. 157, no. 1, pp. 105–132, 1982.

[174] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[175] L. Oneto, N. Navarin, B. Biggio, F. Errica, A. Micheli, F. Scarselli, M. Bianchini, L. Demetrio, P. Bongini, A. Tacchella, and A. Sperduti, "Towards learning trustworthily, automatically, and with guarantees on graphs: An overview," *Neurocomputing*, vol. 493, pp. 217–243, 2022.

[176] P. Bongini, S. Gardini, M. Bianchini, O. Spiga, and N. Niccolai, "Structural bioinformatics survey on disease-inducing missense mutations,"

Journal of Bioinformatics and Computational Biology, p. 2150008, 2021.

[177] P. Bongini, N. Niccolai, A. Trezza, G. Mangiavacchi, A. Santucci, O. Spiga, M. Bianchini, and S. Gardini, "Structural bioinformatic survey of protein-small molecule interfaces delineates the role of glycine in surface pocket formation," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2020.

[178] D. Varrazzo, A. Bernini, O. Spiga, A. Ciutti, S. Chiellini, V. Venditti, L. Bracci, and N. Niccolai, "Three–dimensional computation of atom depth in complex molecular structures," *Bioinformatics*, vol. 21, no. 12, pp. 2856–2860, 2005.

[179] F. Fraternali and L. Cavallo, "Parameter optimized surfaces (pops): analysis of key interactions and conformational changes in the ribosome," *Nucleic Acids Research*, vol. 30, no. 13, pp. 2950–2960, 2002.

[180] H. A. Hussein, A. Borrel, C. Geneix, M. Petitjean, L. Regad, and A.-C. Camproux, "Pockdrug–server: a new web server for predicting pocket druggability on holo and apo proteins," *Nucleic acids research*, vol. 43, no. W1, pp. W436–W442, 2015.

[181] R. H. Horton and A. M. Lucassen, "Recent developments in genetic/genomic medicine," *Clinical Science*, vol. 133, no. 5, pp. 697–708, 2019.

[182] M. J. Landrum, J. M. Lee, M. Benson, G. R. Brown, C. Chao, S. Chitipiralla, B. Gu, J. Hart, D. Hoffman, W. Jang, *et al.*, "Clinvar: improving access to variant interpretations and supporting evidence," *Nucleic acids research*, vol. 46, no. D1, pp. D1062–D1067, 2018.

[183] P. D. Stenson, M. Mort, E. V. Ball, K. Shaw, A. D. Phillips, and D. N. Cooper, "The human gene mutation database: building a comprehensive mutation repository for clinical and molecular genetics, diagnostic testing and personalized genomic medicine," *Human genetics*, vol. 133, no. 1, pp. 1–9, 2014.

[184] E. Capriotti, P. Fariselli, and R. Casadio, "A neural–network–based method for predicting protein stability changes upon single point mutations," *Bioinformatics*, vol. 20, no. suppl_1, pp. i63–i68, 2004.

[185] S. Kulshreshtha, V. Chaudhary, G. K. Goswami, and N. Mathur, "Computational approaches for predicting mutant protein stability," *Journal of computer-aided molecular design*, vol. 30, no. 5, pp. 401–412, 2016.

[186] J. D. Stephenson, R. A. Laskowski, A. Nightingale, M. E. Hurles, and J. M. Thornton, "Varmap: a web tool for mapping genomic coordinates to protein sequence and structure and retrieving protein structural annotations," *Bioinformatics*, vol. 35, no. 22, pp. 4854–4856, 2019.

[187] S. Gardini, S. Furini, A. Santucci, and N. Niccolai, "A structural bioinformatics investigation on protein–dna complexes delineates their modes of interaction," *Molecular BioSystems*, vol. 13, no. 5, pp. 1010–1017, 2017.

[188] H. Hwang, F. Dey, D. Petrey, and B. Honig, "Structure–based prediction of ligand–protein interactions on a genome–wide scale," *Proceedings of the National Academy of Sciences*, vol. 114, no. 52, pp. 13685–13690, 2017.

[189] P. Bongini, A. Trezza, M. Bianchini, O. Spiga, and N. Niccolai, "A possible strategy to fight covid-19: interfering with spike glycoprotein trimerization," *Biochemical and biophysical research communications*, vol. 528, no. 1, pp. 35–38, 2020.

[190] P. Bongini, A. Trezza, M. Bianchini, O. Spiga, and N. Niccolai, "Structural bioinformatics to unveil weaknesses of coronavirus spike glycoprotein stability," in *In Silico Modeling of Drugs Against Coronaviruses*, pp. 203–211, Springer, 2021.

[191] W. Spaan, D. Cavanagh, and M. Horzinek, "Coronaviruses: structure and genome expression," *Journal of General Virology*, vol. 69, no. 12, pp. 2939–2952, 1988.

[192] B. Delmas and H. Laude, "Assembly of coronavirus spike protein into trimers and its role in epitope expression," *Journal of virology*, vol. 64, no. 11, pp. 5367–5375, 1990.

[193] D. Wrapp, N. Wang, K. S. Corbett, J. A. Goldsmith, C.-L. Hsieh, O. Abiona, B. S. Graham, and J. S. McLellan, "Cryo–em structure of

the 2019–ncov spike in the prefusion conformation," *Science*, vol. 367, no. 6483, pp. 1260–1263, 2020.

[194] F. Sievers, A. Wilm, D. Dineen, T. J. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Remmert, J. Söding, *et al.*, "Fast, scalable generation of high-quality protein multiple sequence alignments using clustal omega," *Molecular systems biology*, vol. 7, no. 1, p. 539, 2011.

[195] D. S. Wishart, Y. D. Feunang, A. C. Guo, E. J. lo, A. Marcu, J. R. Grant, T. Sajed, D. Johnson, C. Li, Z. Sayeeda, *et al.*, "Drugbank 5.0: a major update to the drugbank database for 2018," *Nucleic acids research*, vol. 46, no. D1, pp. D1074–D1082, 2018.

[196] J. Mateus, A. Grifoni, A. Tarke, J. Sidney, S. I. Ramirez, J. M. Dan, Z. C. Burger, S. A. Rawlings, D. M. Smith, E. Phillips, *et al.*, "Selective and cross-reactive sars-cov-2 t cell epitopes in unexposed humans," *Science*, vol. 370, no. 6512, pp. 89–94, 2020.

[197] D. L. Wheeler, T. Barrett, D. A. Benson, S. H. Bryant, K. Canese, V. Chetvernin, D. M. Church, M. DiCuccio, R. Edgar, S. Federhen, *et al.*, "Database resources of the national center for biotechnology information," *Nucleic acids research*, vol. 36, no. suppl_1, pp. D13–D21, 2007.

[198] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[199] C. B. Anfinsen, "Principles that govern the folding of protein chains," *Science*, vol. 181, no. 4096, pp. 223–230, 1973.

[200] Z. Li and Y. Yu, "Protein secondary structure prediction using cascaded convolutional and recurrent neural networks," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pp. 2560–2567, 2016.

[201] R. Heffernan, Y. Yang, K. Paliwal, and Y. Zhou, "Capturing non–local interactions by long short-term memory bidirectional recurrent neural networks for improving prediction of protein secondary structure, backbone angles, contact numbers and solvent accessibility," *Bioinformatics*, vol. 33, no. 18, pp. 2842–2849, 2017.

[202] F. M. Pearl, C. Bennett, J. E. Bray, A. P. Harrison, N. Martin, A. Shepherd, I. Sillitoe, J. Thornton, and C. A. Orengo, "The cath database: an extended protein family resource for structural and functional genomics," *Nucleic acids research*, vol. 31, no. 1, pp. 452–455, 2003.

[203] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[204] S. N. Wakap, D. M. Lambert, A. Olry, C. Rodwell, C. Gueydan, V. Lanneau, D. Murphy, Y. Le Cam, and A. Rath, "Estimating cumulative point prevalence of rare diseases: analysis of the orphanet database," *European Journal of Human Genetics*, vol. 28, no. 2, pp. 165–173, 2020.

[205] M. Navaie-Waliser, P. H. Feldman, D. A. Gould, C. Levine, A. N. Kuerbis, and K. Donelan, "When the caregiver needs care: The plight of vulnerable caregivers," *American journal of public health*, vol. 92, no. 3, pp. 409–413, 2002.

[206] E. Palamaro Munsell, R. P. Kilmer, J. R. Cook, and C. L. Reeve, "The effects of caregiver social connections on caregiver, child, and family well-being.," *American Journal of Orthopsychiatry*, vol. 82, no. 1, p. 137, 2012.

[207] F. Guerranti, M. Mannino, F. Baccini, P. Bongini, N. Pancino, A. Visibelli, and S. Marziali, "Caregivermatcher: graph neural networks for connecting caregivers of rare disease patients," *Procedia Computer Science*, vol. 192, pp. 1696–1704, 2021.

[208] L.-Y. Chien, H. Chu, J.-L. Guo, Y.-M. Liao, L.-I. Chang, C.-H. Chen, and K.-R. Chou, "Caregiver support groups in patients with dementia: a meta-analysis," *International journal of geriatric psychiatry*, vol. 26, no. 10, pp. 1089–1098, 2011.

[209] M. Monaci, N. Pancino, P. Andreini, S. Bonechi, P. Bongini, A. Rossi, G. Ciano, G. Giacomini, F. Scarselli, and M. Bianchini, "Deep learning techniques for dragonfly action recognition.," in *ICPRAM*, pp. 562–569, 2020.

[210] S. Bonechi, M. Bianchini, P. Bongini, G. Ciano, G. Giacomini, R. Rosai, L. Tognetti, A. Rossi, and P. Andreini, "Fusion of visual and anamnestic data for the classification of skin lesions with deep learning," in *International Conference on Image Analysis and Processing*, pp. 211–219, Springer, 2019.

[211] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[212] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[213] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[214] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[215] U. Leiter, T. Eigentler, and C. Garbe, "Epidemiology of skin cancer," *Sunlight, vitamin D and skin cancer*, pp. 120–140, 2014.

[216] N. C. Codella, D. Gutman, M. E. Celebi, B. Helba, M. A. Marchetti, S. W. Dusza, A. Kalloo, K. Liopyris, N. Mishra, H. Kittler, *et al.*, "Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (isbi), hosted by the international skin imaging collaboration (isic)," in *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*, pp. 168–172, IEEE, 2018.