universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Learning with Reduced Molecular Graphs“

verfasst von / submitted by

Vojtech Voracek Bc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 645
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Data Science
Betreut von / Supervisor:	Ass.-Prof. Dipl.-Inf. Dr. Nils Morten Kriege

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Nils Morten Kriege, for his invaluable guidance, unwavering support, and insightful feedback throughout the entire process of conducting this work.

I am also deeply thankful to my co-supervisor, Franka Bause, for her constructive suggestions, thoughtful discussions, and continuous encouragement.

I want to extend my thanks to my family for their endless support and for providing me with the opportunity to not only dedicate myself to my thesis but also to engage in household duties.

My heartfelt thanks go to my friends for all the trips, summer camps, weddings, and joyful moments. Without you, I would have finished this thesis much sooner.

I want to convey my deepest gratitude to my future wife and children, the guiding light of my existence.

Abstract

This thesis focuses on the use of machine learning techniques in drug discovery, specifically the application of graph-based techniques to predict molecules' properties. While Graph Neural Networks have shown promise in this field, they face challenges in distinguishing between different types of molecules. One solution to this problem is the Hierarchical Inter-Message Passing scheme, which operates on multiple graphs and allows for the aggregation of information across them.

The thesis introduces a graph learning technique based on the Hierarchical Inter-Message Passing scheme. Our approach leverages reduced graph representations, including the Extended Reduced Graph and the Feature Tree, in addition to the original molecular graph. This allows for a more comprehensive understanding of molecular structure.

Experimental results on benchmark datasets reveal the superiority of the developed model over existing approaches and state-of-the-art techniques. This thesis contributes to advancing the field by providing a powerful model for accurately predicting molecular properties in drug discovery and has implications for accelerating the drug discovery process.

Kurzfassung

Diese Arbeit befasst sich mit dem Einsatz von Techniken des maschinelles Lernens in der Arzneimittelforschung, insbesondere mit der Anwendung graphbasierter Techniken zur Vorhersage der Eigenschaften von Molekülen. Graph Neuronale Netze haben sich in diesem Bereich zwar als vielversprechend erwiesen, stehen aber vor der Herausforderung, zwischen verschiedenen Molekülarten zu unterscheiden. Eine Lösung für dieses Problem ist das hierarchische Inter-Message-Passing-Schema, das mit mehreren Repräsentationen des selben Graphen arbeitet und die Aggregation von Informationen über sie hinweg ermöglicht.

Die Arbeit stellt eine Graphlernmethode vor, die auf dem hierarchischen Inter-Message-Passing-Schema basiert. Unser Ansatz nutzt reduzierte Graphrepräsentationen, einschließlich des Extended Reduced Graphs und des Feature Trees, zusätzlich zum ursprünglichen Molekülgraphen. Dies ermöglicht ein umfassenderes Verständnis der molekularen Struktur.

Experimentelle Ergebnisse anhand von Benchmark-Datensätzen zeigen die Überlegenheit des entwickelten Modells gegenüber bestehenden Ansätzen und modernen Techniken. Diese Arbeit trägt zur Weiterentwicklung des Fachgebiets bei, indem sie ein leistungsstarkes Modell zur präzisen Vorhersage von molekularen Eigenschaften in der Arzneimittelforschung bereitstellt und Auswirkungen auf die Beschleunigung des Arzneimittelentdeckungsprozesses hat.

Contents

List of Tables

List of Figures

1 Introduction

Drug discovery is a very important part of the process of developing new medications and therapies and plays a pivotal role in modern medicine. This process usually consists of many stages, including target identification, lead generation, lead optimization, preclinical testing, and clinical trials.

The early stages of drug discovery can significantly benefit from machine learning techniques [DDJ+22]. The majority of those techniques accept vectors as input. However, due to the complex nature of molecules, which are made up of multiple atoms and chemical bonds, it is not feasible to represent them accurately using vectors. Therefore, researchers have turned to graph-based representations as a more appropriate way to model molecules [SYS+20][JEP+21][BO04].

Various graph representations exist for molecules, including the commonly used approach of representing atoms as nodes and chemical bonds as edges. However, so-called reduced graph models are also available, representing groups of atoms with a single node and capturing their properties using node attributes. These reduced graph models can be beneficial when working with large or complex molecules, as they can simplify the structure and make it easier to analyze [BG11]. Additionally, each model can represent a distinct aspect of molecules so that they can be well-suited for various tasks.

In recent years, there have been significant advancements in machine learning methods for working with graph structures. One such technique is Graph Neural Networks (GNNs), which are well-engineered algorithms that can be applied to graphs annotated with node and edge attributes [SGT+09]. GNNs work by iteratively updating node representations based on information from neighboring nodes. This approach has shown great promise in various applications, including drug discovery research [CAEHP+22] [GSR+17] [DMAI+15]. Nevertheless, studies have revealed that GNNs exhibit certain limitations. These limitations include difficulties in detecting cycles in molecular graphs [Lou20], an inability to distinguish certain types of molecules and challenges in encoding different aspects of molecules [GGG20] [XHLJ19].

As a result, a partial solution to this problem has been introduced in the form of the Hierarchical Inter-Message Passing scheme (HIMP)[FYW20]. This model operates on reduced graph representation along with the original graph. It utilizes an associated Feature Tree[1] of the graph [RD98]. The Feature Tree captures meaningful clusters, such as rings or bridged compounds, represented as nodes in the tree. The model leverages message passing within each graph to learn a comprehensive molecular representation. Additionally, it facilitates bidirectional information exchange between the two graphs for a more complex understanding of the molecular structure. These extensions allow HIMP

[1]The Feature tree is referred to as a Junction tree in [FYW20]

to overcome some limitations of standard GNNs mentioned before and achieve better results. However, there is still room for improvement.

In this thesis, we will explore if the usage of various reduced graphs may be beneficial for property prediction with GNNs. The goal of this work is to develop new graph learning techniques based on HIMP in order to achieve better results than the original HIMP. We extend HIMP to be capable of operating on an arbitrary number of graphs. We will use different reduced graph representations, namely Extended Reduced Graph (ErG) [SWBZ06]and Feature Tree (FT) [RD98].

Finally, we compare the developed extended HIMP variants with state-of-the-art methods.

1.1 Related Work

1.1.1 Overcoming Limitations of Standard GNNs

In [XHLJ19], it was revealed that the conventional GNNs framework is not a universal approximator of functions on graphs. Specifically, it is shown to be equivalent to the 1-dimensional Weisfeiler-Leman test (1-WL) [WL68]. Even though such a network has the potential to be highly effective at distinguishing patterns, it has been demonstrated that real-world networks can encounter difficulties when trying to perform simple tasks like detecting small cycles [CCVB20]. This appears to be a serious problem in predicting molecular properties since molecular properties are largely influenced by their local structure, such as aromatic groups. In the end, this may negatively affect a network's performance.

Multiple attempts to solve this problem have been proposed. Some recent studies have explored alternative architectures that diverge from the traditional convolution-based framework, relying on sophisticated models such as invariant graph networks [MBHSL19a] [MBHSL19b], or relational pooling [MSRR19] [CCVB20]. Another approach is to extend the current framework, to make it equivalent to higher order Weisfeiler-Leman tests [MRF+19].

In contrast to those methods, HIMP scheme offers a simple and efficient solution with minimal additional costs in terms of memory and execution time. Despite its simplicity, HIMP has demonstrated the ability to achieve state-of-the-art performance on multiple datasets [FYW20].

1.1.2 Reduced Graphs Used in GNNs

The integration of reduced graphs in Graph Neural Networks is not widely adopted in current research. Until now, Feature Trees have been exclusively employed for molecule generation using a coarse-to-fine generation procedure, as demonstrated in [JYBJ19] and [JBJ18]. The RG-MPNN model for chemical property prediction and interpretation utilizes a reduced graph representation that is closely related to Extended Reduced Graph representation [KZL+22]. The RG-MPNN model operates on both the original graph and a reduced graph. However, it differs from HIMP in that it does not operate on both

graphs simultaneously. Instead, the process involves three distinct phases. The first phase is a message passing phase on an original graph, followed by a graph reduction in the second phase. Finally, a message passing phase is conducted at the reduced graph level in the third phase.

In a recent study [KO23], a novel model employing multiple Reduced Graphs within the framework of a Graph Neural Network (GNN) was introduced. This model may operate on four distinct graphs, namely, the molecular graph, Feature Tree, Extended Reduced Graph, and the Functional Group Graph [JSL+22]. This model is closely related to our model as it aims to increase the model's performance by utilizing multiple reduced graphs along the molecular graph.

1.1.3 Inter-Message Passing

The concept of information exchange between graphs has been extensively explored in practical applications. Specifically, in the area of deep graph matching, which is studied in [WLLZ18], [LGD+19], and [FLM+20], the goal is to compare two or more graphs and determine their structural similarities. In contrast to our approach, where different representations of the same graph are used to pass messages, deep graph matching involves passing messages between representations of different graphs.

Another field that has gained significant attention is graph pooling, wherein the majority of research efforts are directed toward developing techniques for learning a coarser representation of an input graph, such as DIFFPOOL [**?**], and top_k [GJ22] pooling approach. In this context, inter-message passing is used to allow information exchange between different parts of the graph during the pooling process. This enables the representative node to capture the most important information from the group of nodes and pass it on to the next layer of the network for further processing.

2 Foundations

2.1 Graphs

The theoretical foundations presented in this section are largely taken from books [SP14] and [vS10].

A graph is a mathematical structure representing a set of objects in which pairs of objects may be related. The objects are called nodes (also known as vertices), and each relationship between a pair of ndoes is represented by an edge (also called a link).

Graphs are typically visualized as a set of dots or circles for nodes, which are connected by lines for edges. An example of such visualization may be seen in Figure 2.1.

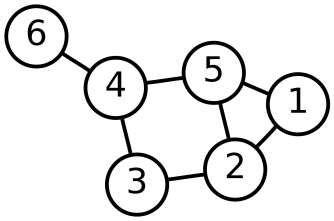

Figure 2.1: An example of a graph with six nodes and seven edges. Taken from [Con23a].

In mathematical notation, we can define a **graph** (simple graph) G as a tuple

$$G = (V, E),$$

where V is a set of **nodes** and E is a set of **edges**. Each edge $e_i \in E$ is an unordered pair of nodes $u_i, v_i \in V$, such that $u_i \neq v_i$.

Additionally, we can define a function $f_v : V \to X$ and a function $f_e : E \to Y$ that assign labels to nodes and edges, respectively.

When edge $e \in E$ connects nodes $u, v \in V$, we say those nodes are to be **adjacent**. It is common to use the notations $V(G)$ and $E(G)$ to refer to the set of nodes and edges of a graph G, respectively.

For a graph G and node $v \in V(G)$, we define the **neighborhood** of the node v as the set of nodes adjacent to v. More formally:

$$N(v) \ = \ \{u \ \in \ V(G) \mid (u, \ v) \ \in \ E(G)\}$$

A **walk** in graph G is an alternating sequence $[v_0, e_1, v_1, e_2, ..., v_{k-1}, e_k, v_k]$ of nodes and edges, where $\forall i : v_i \in V(G)$, and $\forall j : e_j = \{v_{j-1}, v_j\} \in E(G)$. A **trail** is a walk where all edges are distinct. A **path** is a trail in which all nodes are unique as well. A **cycle** is a

trail, where $v_0 = v_k$ and all other nodes are distinct. A **biconnected component** of a graph G is a maximal set of edges such that any two edges in the set lie on a common cycle.

Two nodes $u, v \in V(G)$ are said to be **connected** if there exists a path in G from u to v. The graph G is called connected if every pair of nodes in the graph is connected. A graph G that is both connected and does not contain any cycle is commonly referred to as a **tree**. Figure 2.2 shows a graph that is a tree.

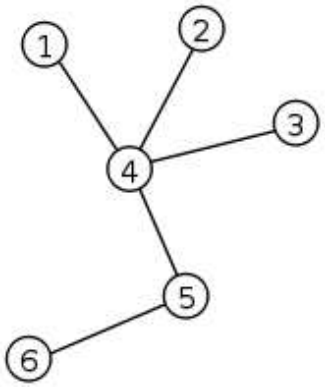

Figure 2.2: An example of a tree. Taken from [Con23c].

We say that graphs G and T are **isomorphic**, denoted as $G \cong T$, if there exists a bijection between nodes of G and T:

$$f : V(G) \to V(T),$$

such that any two vertices $u, v \in V(G)$ are adjacent in G if and only if $f(u), f(v) \in V(T)$ are adjacent in T. We refer to f as an **isomorphism**.

2.1.1 Molecular Graphs

Molecular graphs are a specific type of graphs that are used to represent the structure of molecules [Tri83]. In a molecular graph, the atoms of a molecule are represented as nodes, and the bonds between the atoms are represented as edges.

Each node in a molecular graph is labeled with the chemical symbol of the corresponding atom. For example, a node representing a nitrogen atom would be labeled with the symbol "N", while a node representing an oxygen atom would be labeled with the symbol "O". It should be noted that when displaying a molecular graph, carbon atoms are typically depicted as unlabeled nodes, as shown in Figure 2.3.

Edges in a molecular graph are labeled with the type of bond between the corresponding atoms. The most common types of bonds in organic chemistry are single bonds, double bonds, and triple bonds, which are usually represented by the symbols "-", "=", and "#", respectively. Other types of bonds, such as aromatic and hydrogen bonds, can be represented using different symbols.

Formally, we define a molecular graph as a graph $G = (V, E)$, where V represents the set of atoms in the molecule and E represents the set of bonds between these atoms, together with two mapping functions f_v and f_e. The function $f_v : V \to X$ assigns chemical symbols to the nodes, where X represents the set of all chemical symbols. Similarly, the function $f_e : E \to Y$ assigns bond types to the edges, with Y representing the set of all bond type symbols.

In order to simplify the handling of molecular graphs, it is common to use graphs that depict only the molecular skeleton without the inclusion of hydrogen atoms and their corresponding bonds. This is done because the omission of the hydrogen atoms and their corresponding bonds, specifically the bonds connecting hydrogen atoms with other atoms in the molecule, in most cases, cannot be the cause of any ambiguity. Furthermore, the heavier atoms have a greater impact on the properties and behavior of the molecule [Spi64].

The molecular graph provides a simplified representation of a molecule by depicting only its constitution and disregarding other structural features such as geometry, stereochemistry, and chirality. Despite this simplification, a molecular graph can still be a very useful tool for predicting a molecule's physical and chemical properties.

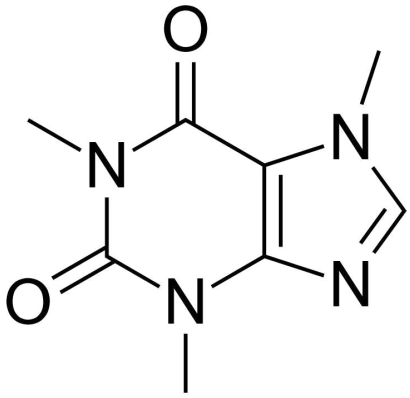

Figure 2.3: A molecular graph of caffeine [Con23b].

2.2 Reduced Graphs

A reduced graph is a simplified representation of a molecular graph where each node represents a set of atoms of an original graph. An edge is present between two nodes of a reduced graph u, v if, in the original graph, there is a bond between any atom contained in u and any atom contained in v. The properties of a particular group of atoms are represented by a node attribute of the reduced graph [GDH+91].

More formally, if G' is a reduced graph of a molecular graph G, then:

$$G^{'} = (V^{'}, E^{'})$$
$$V^{'} = \{v_1^{'}, ..., v_m^{'}\}$$
$$E^{'} = \{e_1^{'}, ..., e_m^{'}\},$$

where:

$$\forall i : v_i^{'} \subseteq V(G)$$
$$(v_i^{'}, v_j^{'}) \in E^{'} \iff \exists v_i \in v_i^{'}, v_j \in v_j^{'} : (v_i, v_j) \in E(G)$$

In addition, we define the mapping function $f_v^{'} : V^{'} \to Z$ that maps a set of atoms of an original molecular graph to a node attribute. An example of a reduced graph is captured in Figure 2.4

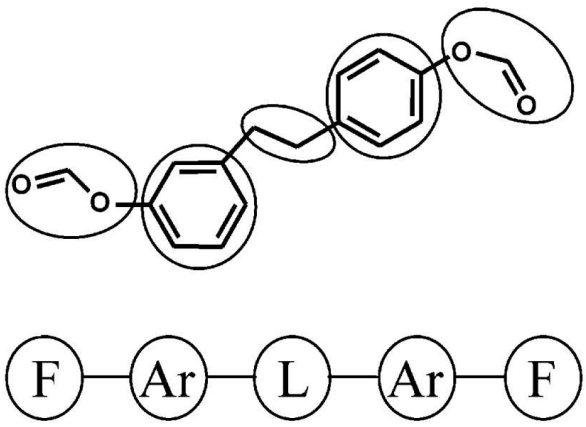

Figure 2.4: The molecular graph is shown in the upper part of the figure. The bottom part shows its reduced graph, with F, Ar, and L being the node attributes. Each circled group of atoms of the original graph is mapped to the corresponding node of the reduced graph below it. The figure was taken from [GWB03].

Reduced graphs provide a summary of a molecule's features while preserving the topology between them. This enables molecules to be generalized based on the identification of properties associated with individual atoms or groups of atoms, along with their respective topology. This can be useful for various applications, including drug discovery or molecular similarity search [TSS92] [GWB03].

Several graph reduction techniques have been developed, and the most effective scheme is expected to vary depending on the specific application. In the following two subsections, we will introduce two types of reduced graphs that we will employ in our experiments.

2.2.1 Extended Reduced Graph

The first type of reduced graph representation presented in this work is the Extended Reduced Graph (ErG) method, which was introduced by Stiefl et al. in [SWBZ06]. This method is implemented within RDKit software [Lan16].

It combines a reduced graph with additional nodes and edges that represent the pharmacophoric features of the molecule. The reduced graph captures the essential topological relationships between atoms in the molecule. At the same time, the additional nodes and edges encode the pharmacophoric features that are important for the molecule's biological activity.

The pharmacophoric features can include specific chemical groups or patterns that are known to be important for the molecule's activity, such as hydrogen bond acceptors or donors, aromatic rings, or hydrophobic groups.

Formally, ErG is not a standard reduced graph since it introduces new artificial atoms, which violates the following condition of reduced graphs: $\forall i : v_i^{'} \subseteq V(G)$. Nevertheless, for the purposes of this study, it holds no significance.

The process of constructing an ErG involves the application of a predefined set of pharmacophoric rules to an original molecular graph. A high-level algorithm outlining the steps involved in this process is presented below:

1. Atom charging Initially, the atoms are assigned formal charges to represent the molecule under physiological conditions.

2. H-bond property assignment The next step involves assigning hydrogen-bond donor and acceptor properties to the atoms. Any atoms that exhibit donor or acceptor features are labeled with a so-called flip-flop flag, which is taken into account explicitly before generating descriptors.

3. Endcap identification The next step involves identifying the "endcap" groups, which are lateral hydrophobic features made up of three atoms (such as isopropyl). These groups are important since they usually contribute significantly to the shape and size of the molecule. In addition, thioethers, which are composed of only two atoms, are also treated as endcap groups.

Notably, some terminal non-hydrogen atoms may be excluded from the endcap group. This approach results in a more general representation of the molecule while reducing the weight given to these atoms in the reduced graph.

4. Ring conversion In the fourth step, the ring systems are converted into their abstract forms. The abstraction of the ring systems consists of the following steps:

1. For each identified ring system, create an artificial atom called a centroid. An identified ring system is a ring system that contains fewer than eight atoms. Assign a corresponding flag to the centroid (Ar for aromatic ring systems, Hf for hydrophobic group).

2. Keep all atoms that belong to multiple identified rings, and for each of these atoms, create a bond between the atom and the centroid of every ring the atom is a part of.

3. Retain all atoms of the ring that are either substituted or are assigned a charge, H-bond donor, or H-bond acceptor flag. Create a bond between each of these atoms and the centroid.

4. Remove all other ring atoms. Keep all bonds between atoms retained in the two previous steps.

It is important to note that this step may lead to the resulting ErG being not connected. It is also possible that some atoms of the molecule may be omitted during this process. Since only rings of length smaller than eight are identified, atoms that fulfill the following three conditions are not represented in any way in the ErG:

- The atom is a member of a ring of length at least eight
- An atom is connected to only two other atoms in the molecular graph
- No property has been assigned to the atom in the previous steps

The whole process of conversion of the molecular graph into Extended Reduced Graph is shown in Figure 2.5.

2.2.2 Feature Tree

The Feature Tree (FT) method [RD98] is the second type of reduced graph representation used in this work. The Feature Tree has been implemented within the Pytorch geometric software [FL19].

A feature tree is a hierarchical data structure that represents a molecular structure as a tree, where nodes correspond to molecular fragments and edges represent chemical bonds between those fragments. Each node in the tree contains a feature that is obtained from the corresponding molecule fragment. As a result, the feature tree provides a rough approximation of the molecular structural formula.

Using a tree data structure instead of a general graph data structure is motivated by algorithmic complexity and efficiency considerations. Trees are a specific type of graph with a hierarchical structure, making certain operations on them more efficient than on general graphs. This becomes particularly significant when processing graphs using GNNs. The original paper leveraged this aspect specifically for solving a graph matching problem.

In a Feature Tree, a node is used to represent a set of atoms within a molecule that are connected in the molecular graph. Each atom in the molecule is associated with at least one node of FT. Nodes of FT that share atoms or contain atoms connected in the molecular graph are connected. It is important to note that to ensure a well-defined tree, the sets of atoms must be carefully chosen to prevent any cycles in the resulting FT.

A generation of a Feature Tree from a molecule consists of several steps. Below is a presented algorithm at a high level, which outlines the steps involved in this process.

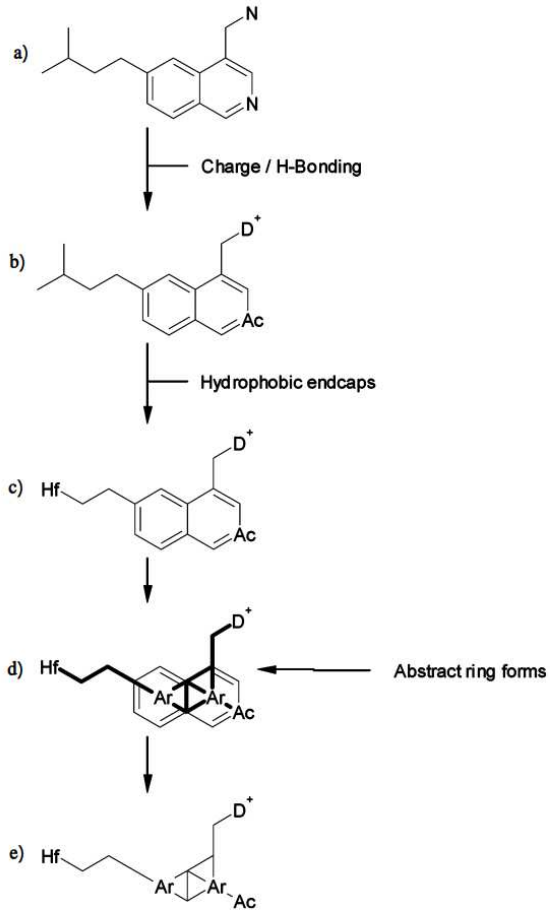

Figure 2.5: Conversion of the molecular graph of a sample molecule into ErG a) molecular graph, b) a graph after the first two steps of the conversion procedure, D is H-bond donor flag, Ac is H-bond acceptor flag, and + is a positive charge, c) A graph after endcap identification, Hf is a hydrophobic group, d) A graph in a ring conversion phase, just before removal of ring atoms without flags, Ar is an aromatic ring system flag, e) A resulting ErG of a sample molecule. The figure was taken from [SWBZ06].

1. Ring identification First, all ring systems of the molecular graph are found by biconnected component algorithm [CLRS01].

2. Cycle-graph construction In the next step, a so-called cycle-graph is constructed. Initially, all ring systems are broken down into a set of rings as follows: For every atom within the ring system, the set of cycles with minimal length containing the atom is identified. The minimal length cycles for an atom are obtained by a breadth-first-search algorithm closely related to the biconnected component algorithm [CLRS01].

These cycles are then gathered in a set of initial cycles, which form the nodes of the

cycle-graph. If two rings share more than two atoms, they are merged into one ring as they form bridged compounds. To connect the nodes in a cycle-graph, an edge is added between each pair of nodes whose corresponding cycles have at least one atom in common.

3. Cycle-graph to tree conversion If the cycle-graph contains cycles, the cycle-graph's nodes are decomposed using the biconnected component algorithm. It is worth noting that all cycles within the cycle-graph are precisely covered by the biconnected components. As a result, we can represent each biconnected component by a single node to obtain a tree structure. Finally, we merge the cycles represented by nodes of the biconnected components in the cycle-graph, so we can obtain a tree that serves as the feature tree for the ring system.

4. Acyclic part treatment In this step, we deal with all nodes of a molecular graph that are not part of any ring system. For the acyclic part of the molecule, each edge that is not contained in any ring system is treated as a separate node in the resulting Feature Tree. Those nodes are assigned a bond feature. Each bond node represents the two nodes of a molecular graph connected with the corresponding edge. Again, edges are added between those nodes of FT that share at least one node of a molecular graph. Finally, note that hydrogen atoms and bonds with them are considered explicitly.

5. Zero nodes As previously stated, edges are added between nodes of FT that contain connected atoms in the molecular graph. During this process, cycles of size three may appear, as depicted in Figure 2.6. These cycles have to be eliminated in order to obtain a tree structure. It is done by introducing so-called zero nodes. These nodes represent an empty set of atoms. For each of those cycles, one zero node is created. All nodes of the cycle are then connected to a corresponding zero node with an edge. Finally, all edges that form the cycle are removed.

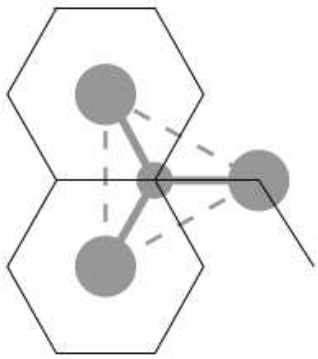

Figure 2.6: The fifth part of a FT generation process. The cycle of length three (dashed grey lines) is eliminated by introducing a zero node (smaller grey disk). Solid grey lines indicate newly introduced edges. The figure was taken from [RD98].

6. Singleton nodes Bonds, rings, and bridged compounds are called clusters. For each node of the molecular graph that lies inside more than two clusters, a separate node in the resulting Feature Tree is created. These nodes are assigned the singleton flag.

This procedure results in a unique Feature Tree for a particular molecule. Each node of a FT contains one of the following four features: bond, ring, bridged compound, singleton. An example of a Feature Tree is depicted in Figure 2.7.

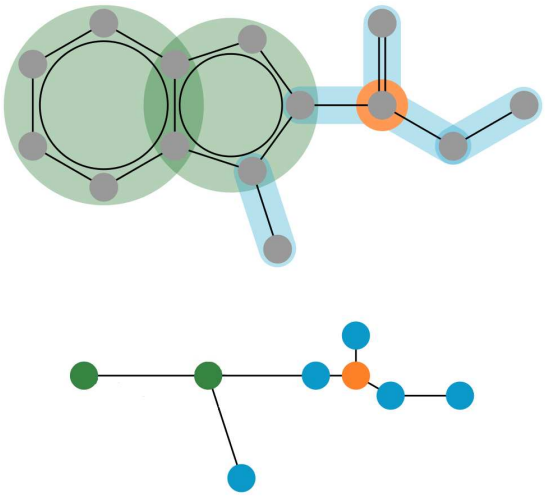

Figure 2.7: The upper part of the figure shows a molecular graph of a sample molecule. Its Feature Tree is shown in the bottom part. Colored parts overlaid on the molecular structure indicate which atoms are joined to one node. Color encodes features of nodes of FT. Orange color refers to singleton, red to ring, and blue to bond. The figure was taken from [FYW20].

2.3 Neural Networks

This section provides an overview of neural networks and their architecture based on the relevant literature [Hay09, Nie15, Agg18, HDBDJ14].

A Neural Network (NN) is a machine learning algorithm that is inspired by the biological neural networks found in human brains. It consists of neurons organized into layers, with each neuron receiving input from other neurons and applying a nonlinear transformation to produce an output vector, which is then passed to the next layer of neurons. The output of the final layer ($\hat{y}$) represents the network's prediction or classification of the input data (x).

The connections between neurons in the network are characterized by a set of parameters (θ), which are learned through a training process that involves updating these parameters

based on the difference between the network's output and the desired output. This process typically uses a backpropagation algorithm for supervised learning.

From a mathematical standpoint, the Neural Network can be defined as a function $f(x, \theta) = \hat{y}$, where x represents the input data, θ denotes the parameters to be learned during the training process, and $\hat{y}$ is the output of the network.

Neural networks can be used for regression or classification and have been widely used in both industry and research. The capabilities of NNs were estimated by the Universal Approximation Theorem established by Cybenko [Cyb89] and Hornik [Hor91]. The Universal Approximation Theorem states that a neural network with a single hidden layer can approximate any continuous function on a compact input domain with arbitrary accuracy.

In the following subsections, we introduce key concepts of neural networks.

2.3.1 Neuron

A neuron is a fundamental unit of a NN inspired by the biological neuron in the human brain. It is a function f_j associated with a weight vector $w_j = [w_{j,1}, ..., w_{j,d}] \in \mathbb{R}^{d \times 1}$ that receives an input vector $= [x_1, ..., x_d] \in \mathbb{R}^{d \times 1}$ from other neurons or external sources and computes a weighted sum of those inputs. This weighted sum is then passed through an activation function ϕ, producing an output value $v_j \in \mathbb{R}$. In mathematical notation, the output of a neuron is computed as follows:

$$v_j = f_j(\mathbf{x}) = \phi\big(\sum_{i=1}^{d} x_i w_{j,i}\big) = \phi(\mathbf{x}^T \mathbf{w_j})$$

It is important to note that in neural networks, a bias vector is typically used along with a weight matrix. In order to incorporate the bias, we add a bias dimension with a constant of 1 to all input vectors and extend the weight matrix by 1 column to create a bias column. By appending ones to all input vectors, we only have to learn a single weight matrix that holds both the weights and the biases.

Figure 2.8 shows how a neuron works.

2.3.2 Activation Function

Activation functions are essential in neural networks to prevent linearity by allowing data to pass through nonlinear functions since, without them, the data would only go through linear functions ($x^T w_j$), resulting in a linear function composition. The output would then be the result of a linear function regardless of the number of layers it passes through.

Multiple activation functions can be considered, including Sigmoid, Tahn, or Step functions. However, in this work, we exclusively leverage the ReLU function for our purposes.

ReLU ReLU outputs a value between 0 and infinity and always has a gradient of 1 for positive values of its argument. This function is extremely simple to compute. ReLU is

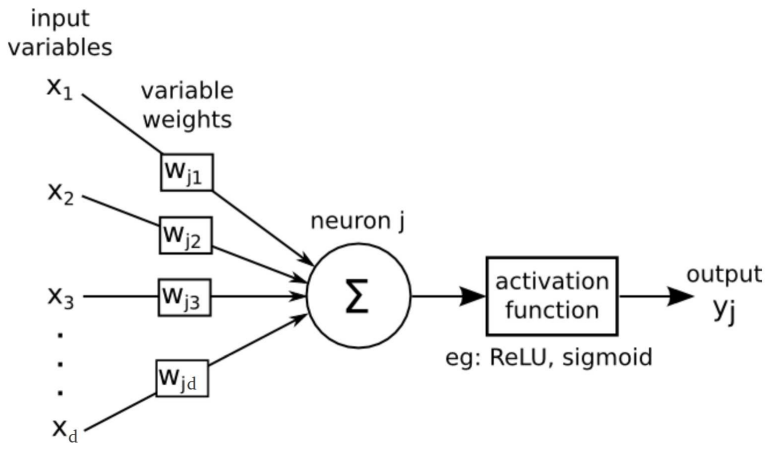

Figure 2.8: A basic scheme of an artificial neuron where $\Sigma = \mathbf{x}^T\mathbf{w_j}$. The figure was taken from [con20].

differentiable everywhere except 0, where the derivative is usually defined as 0.

$$\phi(x) = \max(0, x)$$

Figure 2.9 presents a ReLU function.

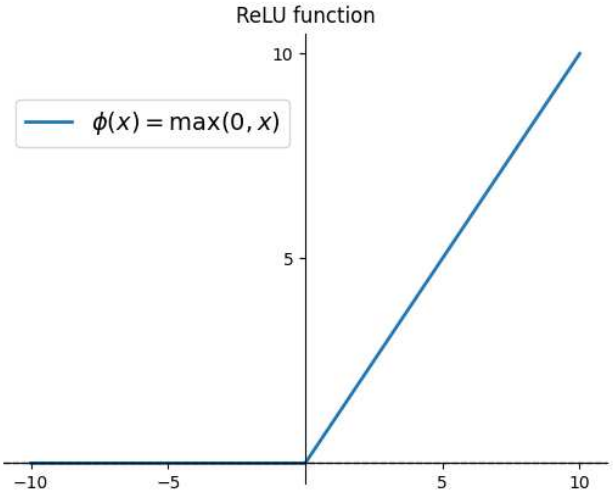

Figure 2.9: A graph of ReLU activation function.

2.3.3 Multilayer Perceptron

A Multilayer perceptron (MLP) is the most basic architecture of NN. MLP is a structure composed of several layers of neurons. The neurons in one layer are connected only to the neurons in the immediately preceding and following layers. The output of a neuron of a certain layer becomes the input of a neuron of the next layer. The **input layer** receives external data, and the **output layer** produces the final result, while zero or more **hidden layers** exist in between. There exist many NN architectures, including convolution NNs, recurrent NNs, or Graph Neural Networks. However, in this subsection, we present only this very basic architecture called MLP. The elementary NN architecture is shown in Figure 2.10.

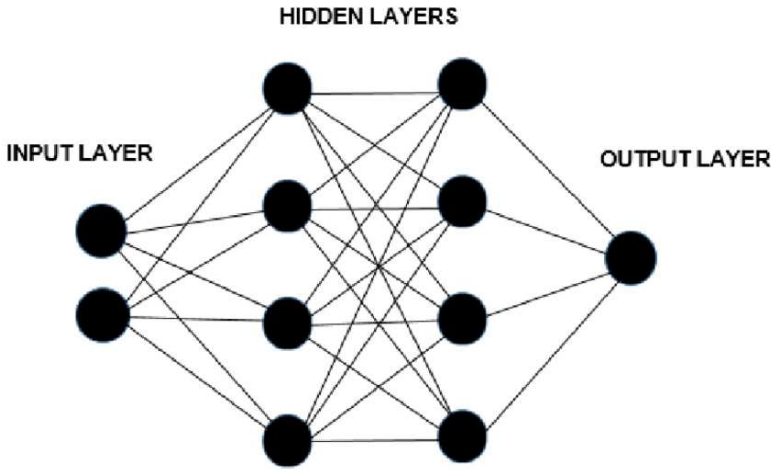

Figure 2.10: A neural network with two input variables, one output variable, and two hidden layers [Bar14].

The mathematical notation of a NN with L hidden layers can be summarized as follows:
For the input layer:

$$v^{(0)} = x,$$

where $x \in \mathbb{R}^{d \times 1}$ is an input vector of dimension d.
For $l = 1, 2, ..., L-1$ (hidden layers)

$$z^{(l)} = W^{(l)} v^{(l-1)}$$

$$v^{(l)} = \phi(z^{(l)}),$$

with $W^{(l)}$ being the weight matrix for layer l composed of the weight vectors $[w_1^{(l)}, ..., w_{h^{(l)}}^{(l)}]^T \in \mathbb{R}^{h^{(l)} \times h^{(l-1)}}$, where $h^{(l)}$ is the number of neurons in layer l and $\phi(z^{(l)})$ is a shortcut for $[\phi(z_1^{(l)}), ..., \phi(z_{h^{(l)}}^{(l)})]$.

For the output layer:

$$z^{(L)} = W^{(L)} v^{(L-1)}$$

$$\hat{y} = f(x, \theta) = \psi(z^{(L)}),$$

where ψ stands for an output layer activation and $\theta = \{W^{(1)}, ..., W^{(L)}\}$ denotes the set of parameters of NN. $\hat{y} \in \mathbb{R}^o$ stands for the output vector of NN, and o is a number of output variables. This implies that $W^{(L)} \in \mathbb{R}^{o \times h^{(L-1)}}$.

Batching

It can be seen that this neural network architecture may process multiple data points simultaneously. This is known as batch processing. We can consider $x \in \mathbb{R}^{d \times B}$, where B is the number of data points, known as a **batch size**. Since activation functions are still applied element-wise, $z^{(l)}, v^{(l)}$ are vectors from $\mathbb{R}^{h^{(l)} \times B}$, and the output is a matrix $y \in \mathbb{R}^{o \times B}$. So the i-th column of the output matrix y is the output vector of the i-th input vector stored in the i-th column of the input matrix x.

This ability to process multiple data points simultaneously is one of the big advantages of neural network architecture since it can be done very fast by utilizing a graphics processing unit (GPU). This allows neural networks to scale up and handle large datasets efficiently.

Regularization

Overfitting is a common problem in machine learning where a model is trained to fit the training data too closely, resulting in poor performance on new and unseen data. To address this problem, regularization techniques have been developed to prevent the model from becoming too complex and overfitting the training data. Regularization techniques are used to improve the generalization of a model and prevent it from memorizing the training data, leading to better performance on new data.

One such regularization technique is **batch normalization** [IS15], which normalizes input to a layer for each batch of data in the training process. The normalized values are then transformed using two learned parameters, scale γ and shift β, before being passed on to the next layer. This technique reduces the internal covariate shift and helps the model to converge faster and generalize better.

Mathematically, batch normalization can be expressed as follows:

$$y_{bn} = \frac{x_{bn} - \mathrm{E}[x_{bn}]}{\sqrt{\mathrm{VAR}[x_{bn}] + \epsilon_{bn}}} * \gamma + \beta,$$

The batch normalization layer takes matrix $x_{bn} \in \mathbb{R}^{h \times B}$ as input and produces matrix $y_{bn} \in \mathbb{R}^{h \times B}$ as output. $\mathrm{E}[x_{bn}]$ and $\mathrm{VAR}[x_{bn}]$ denote mean and standard-deviation per-dimension over the batch. The $*$ operator applies element-wise multiplication (elements in the i-th row of a matrix are multiplied by γ_i). ϵ_{bn} is a user-defined parameter. Finally, $\gamma \in \mathbb{R}^h$ and $\beta \in \mathbb{R}^h$ are learnable parameter vectors.

Another widely used regularization technique is a **dropout** [HSK+12]. During training, dropout randomly drops out a portion of neurons in a given layer, where each neuron has a certain probability p of being dropped out. This forces the model to learn more robust

features and prevents it from relying too heavily on any one feature or set of features. Dropout has been shown to be effective in reducing overfitting in deep neural networks and improving their generalization performance.

2.3.4 Learning

After choosing the architecture of the network, the next step is to learn its parameters (weight matrices $W^{(l)}$ for $l = 1, 2, ..., L$ in the basic scenario) from the training data. This is achieved by minimizing a loss function using a specific optimization algorithm. This training process runs in **epochs** wherein each epoch, the training dataset is fed to the NN model, the loss is calculated, and the model's parameters are adjusted to minimize this loss.

Loss function

In a neural network, the choice of a loss function highly depends on the task being performed. The loss function is a measure of how well the neural network is performing at a given task. It is defined as a function $\mathcal{L}(\theta)$ that takes the neural network's output ($\hat{y}$) and the corresponding input's true known value (y) and returns a scalar value representing the difference between them.

The goal of training the neural network is to find the set of parameters that minimizes the value of the loss function over the entire training dataset.

Even though multiple loss functions are used in Neural Networks, we will use only two of them in this work. The first is BCE with logits loss (BCEL) [PGM+19]. This loss function combines the sigmoid function and the Binary Cross Entropy and, for the i-th data point of the batch and j-th output variable, is defined as follows:

$$\mathcal{L}_{BCEL}(\theta)_{i,j} = -\left[y_{i,j} \cdot \log \sigma(\hat{y}_{i,j}) + (1 - y_{i,j}) \cdot \log(1 - \sigma(\hat{y}_{i,j}))\right],$$

where σ denotes the sigmoid function and subscript i, j indicates the i-th row and j-th column of a matrix.

The second presented loss function is Mean absolute error (MAE). This function is defined only for one-dimensional output for a data point as follows:

$$\mathcal{L}_{MAE}(\theta) = \frac{1}{B} \sum_{i=1}^{B} |\hat{y} - y|,$$

where B denotes the batch size.

Optimization Algorithm

Optimization algorithms are used in NNs to find the optimal values of the parameters that minimize our chosen loss function. Optimization algorithms consider the loss function as a stochastic function of the θ parameters, as it can generate different outputs for the same values of θ parameters when using different data. Typically, we use iterative stochastic

algorithms that rely on computing gradients of the objective function with respect to the model parameters. Parameters are updated using this general formula:

$$\theta_{t+1} = \theta_t - \alpha \cdot g_t,$$

where θ_t, and θ_{t+1} denote parameters at time t, and $t+1$ respectively. g_t is a term that is based on the gradient of a loss function $\mathcal{L}(\theta)$ with respect to θ at time t. Finally, α stands for a so-called **learning rate**.

The learning rate is a crucial hyperparameter of the optimization algorithms. It determines how quickly the model learns from the data and how quickly it updates the model parameters. If the learning rate is too small, the model takes a long time to learn, while if it is too large, the model may fail to converge to the optimal solution or even diverge.

One of the most widely used optimization algorithms in neural networks is the **ADAM** (adaptive momentum estimation) [KB15] optimizer. It computes individual adaptive learning rates for each parameter from estimates of the first and second moments of the gradients of the loss function.

The pseudocode of ADAM is given in Algorithm 1. In the algorithm, $g_t = \nabla_\theta \mathcal{L}_t(\theta_t)$ denotes the gradient i.e. the vector of partial derivatives of $\mathcal{L}_t$, with respect to θ evaluated at time t, with $\mathcal{L}_t$ denoting the realization of the stochastic function $\mathcal{L}$ at time t. Finally, $\mathbf{0}$ stands for the zero vector.

Algorithm 1 Adam optimization algorithm. g_t^2 denotes the element-wise square: $g_t \odot g_t$. All other operations on vectors are also element-wise. β_1^t and β_2^t indicate β_1 and β_2 to the power of t.

Require: $\alpha > 0$: learning rate
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for the moment estimates
Require: $\mathcal{L}(\theta)$: stochastic loss function
Require: θ_0: initial parameter vector
  $m_0 \leftarrow \mathbf{0}$ (initialize the first moment vector)
  $v_0 \leftarrow \mathbf{0}$ (initialize the second moment vector)
  $t \leftarrow 0$ (initialize the timestamp)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta \mathcal{L}_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic loss function at time t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (Compute corrected first moment estimate)
    $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (Compute corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon_A}$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Backpropagation

During the process of learning the parameters of a Neural Network, optimization algorithms need to calculate the gradient of the loss function with respect to all the parameters. An efficient method of computing these partial derivatives can significantly enhance the speed of the learning process.

The **backpropagation** algorithm is a highly efficient way of computing these partial derivatives using the Leibniz chain rule. Backpropagation computes the gradient of the loss function with respect to the parameters of the network for a single input-output example ($\{x, y\}$), layer by layer, starting from the output layer to avoid repetitive calculations of intermediate terms in the chain rule.

We can write the overall NN as a combination of function composition and matrix multiplication (consider the basic MLP architecture introduced in 2.3.3). Note that every function is differentiable:

$$\hat{y} = f(x, \theta) = \psi\Bigg(W^{(L)}\phi\Big(W^{(L-1)}...\phi\big(W^{(1)}x\big)...\Big)\Bigg)$$

For every input-output pair ($\{x, y\}$) in the training set, a loss of the model on that pair is defined as:

$$\mathcal{L}(y, \hat{y}) = \mathcal{L}\Bigg(y, \psi\Big(W^{(L)}\phi\Big(W^{(L-1)}...\phi\big(W^{(1)}x\big)...\Big)\Big)\Bigg)$$

We can initially compute the derivatives with respect to the parameters of the output layer. For the output layer:

$$\text{Compute error signal: } \delta^{(L)} = \left[\frac{\partial \mathcal{L}}{\partial \hat{y}_1}\phi'(z_1^{(L)}), ..., \frac{\partial \mathcal{L}}{\partial \hat{y}_o}\phi'(z_o^{(L)})\right]^T \in \mathbb{R}^{o\times 1}$$

$$\text{Compute gradient } : \nabla_{W^{(L)}}\mathcal{L} = \delta^{(L)}v^{(L-1)T}$$

We proceed by propagating the error signal backward through the hidden layers of the neural network and computing the gradients of the loss function with respect to the parameters in each layer. For each layer, we utilize the error signal from the next layer, calculated in the previous step, to compute the error signal of the current layer.

For each hidden layer $l = L-1, L-2, ..., 1$:

$$\text{Compute error signal } : \delta^{(l)} = \phi'(z^{(l)}) \odot (W^{(l+1)T}\delta^{(l+1)})$$

$$\text{Compute gradient } : \nabla_{W^{(l)}}\mathcal{L} = \delta^{(l)}v^{(l-1)T}$$

This process significantly accelerates the computation of derivatives with respect to all parameters of the Neural Network, resulting in a substantial speed-up of the entire learning process.

2.3.5 Performance measure

Once a Neural Network model has been trained, it is crucial to assess its effectiveness and accuracy in making predictions or classifications. The performance measures provide quantitative evaluations of the model's performance, indicating how it performs on a specific task.

Although there are numerous performance measures available for NNs, we will use only a few of them. For regression tasks, we will utilize the Mean absolute error (MAE) metric, which was introduced in Subsection 2.3.4.

In the case of classification tasks, we will employ two key metrics: The Area Under the Receiver Operating Characteristic (AUC-ROC) [McC89] and The Area Under the Precision Recall Curve (AUC-PRC) [SYZ15]. These metrics assess the performance of the model by measuring the area under the respective curves, ranging from 0 to 1. These metrics are particularly useful as they are robust against imbalanced datasets. Higher values of these metrics indicate superior classifier performance.

The ROC curve plots the true positive rate against the false positive rate as the classification threshold is adjusted. The area under this curve may be calculated as follows:

$$\text{AUC-ROC} = 1 - \frac{1}{|N||P|} \sum_{i:y_i=N} \sum_{j:y_j=P} [[\hat{y}_i \leq \hat{y}_j]],$$

where $|N|$ and $|P|$ represent the number of samples from class N and P, respectively.$[[\cdot]]$ is a function that returns 1 if the argument is true and 0 otherwise, $\hat{y}_i$ is the output of NN representing the probability that the i-th data point belongs to the class P.

The PRC curve is obtained by plotting the precision values against corresponding recall values at various classification thresholds. The area under this curve can be calculated using the following formula:

$$\text{AUC-PRC} = \frac{1}{|P|} \sum_{i:y_i=P} \frac{\sum_{j:y_j=P}[[\hat{y}_i \leq \hat{y}_j]]}{\sum_{j=0}^{n}[[\hat{y}_i \leq \hat{y}_j]]}$$

2.4 Graph Neural Networks

With the growing importance of graph-structured data in various fields, including social network analysis, drug discovery, and recommendation systems, there has been a surge of interest in developing a Neural Network variant for processing data that can be represented as graphs.

This has led to the development of Graph Neural Networks (GNNs) [SGT+09], a type of Neural Network designed to work with graph data. GNNs operate directly on the graph

structure and use message-passing algorithm [GSR+17] to propagate information across nodes and edges. In recent years, GNNs have shown promising results on a variety of tasks, including social network analysis [WLX+20], drug discovery [CAEHP+22, DMAI+15], and many other research areas [DKZ+17], making them an attractive tool for graph-based machine learning applications.

Graph Neural Networks operate on a graph $G = (V, E)$ associated with functions f_v and f_e that assign a label to each node and edge, respectively. First, each node u is transformed into a so-called **node embedding** $x_u \in \mathbb{R}^{d_x}$ and each edge $e = (u, v)$ is transformed into a **edge embedding** $e_{(u,v)} \in \mathbb{R}^{d_e}$. Nodes and edges are transformed into their embeddings based on their label.

In the initial stage, each node and edge is usually assigned a random initial embedding vector. This random initialization plays a crucial role in breaking symmetry among nodes and edges, enabling the model to explore diverse representations and prevent any initial bias towards specific nodes or edges. Consequently, the model can efficiently learn meaningful representations during training, ensuring improved transferability across various graphs or datasets.

Then, a series of so-called message-passing steps are performed to update the embeddings for each node and edge based on the embeddings of their neighbors. This process is repeated for a fixed number of iterations or until a stable embedding is reached. It is important to note that the process is permutation-invariant, which means that GNNs produce the same output for two isomorphic graphs.

2.4.1 Tasks

There are three general types of prediction tasks on graphs that GNNs can perform: edge-level, node-level, and graph-level.

For a node-level task, we predict the role of each node within a graph. The goal is to learn an embedding x_u of a node u such that u's label $l_u = f_v(u)$ can be predicted by GNN as $l_u = f(x_u)$.

In an edge-level task, we want to predict a property of each edge of the graph. This is done by learning an embedding $e_{u,v}$ of an edge (u, v) such GNN can predict the label of this edge termed $l_{(u,v)} = f_e((u, v))$ as $l_{(u,v)} = f(e_{(u,v)})$.

Finally, for a graph-level task, we want to predict the property of an entire graph G. In other words, we want to learn a **graph embedding** h_G for a graph G that helps predict the label of an entire graph $y_G = f(h_G)$. The label h_G is typically obtained from embeddings of the nodes of a graph G, e.g., by taking element-wise sum, mean, or maximum from node embeddings. In this work, we focus exclusively on graph-level tasks.

2.4.2 GNN architecture

Graph Neural Networks follow the neighborhood aggregation strategy implemented as the message-passing layer that is responsible for information exchange between the nodes and their neighbors in a graph. For graph-level tasks, within a message-passing layer, each node aggregates information from its neighboring nodes and the edges that connect

them by gathering their respective embeddings. These embeddings are then aggregated and combined with the node's current embedding to form an updated representation of the node.

Using mathematical notation, the l-th layer of GNN updates embedding of node v as follows:

$$m_v^{(l)} = \text{AGGREGATE}_{\theta_1^{(l)}}\left(\left\{\!\!\left\{\left(x_w^{(l-1)}, e_{(w,v)}^{(l-1)}\right)\right\}\!\!\right\}_{w \in N(v)}\right)$$

$$x_v^{(l)} = \text{UPDATE}_{\theta_2^{(l)}}\left(x_v^{(l-1)}, m_v^{(l)}\right),$$

where $x_v^{(l)}$ and $e_{(w,v)}^{(l-1)}$ represent the embeddings of a node and an edge, respectively, in the l-th layer. $\{\!\{\cdot\}\!\}$ holds for a multiset and $N(v)$ indicates the neighbourhood of a node v. $\theta_1^{(l)}$ and $\theta_2^{(l)}$ denote the trainable parameters of the AGGREGATE and UPDATE functions, respectively. It should be noted that these parameters are different in each layer.

For a graph-level task, after L layers, the graph representation vector is obtained as an output of the READOUT function:

$$h_G = \text{READOUT}\left(\left\{\!\!\left\{x_v^{(L)} | v \in V(G)\right\}\!\!\right\}\right)$$

The choice of AGGREGATE, UPDATE, and READOUT functions is crucial as it specifies a concrete GNN architecture. A great number of architectures have been proposed, including GraphSage [HYL17], Graph Convolutional Networks (GCN) [KW17], or Graph Attention Networks (GAT) [VCC+18].

In this work, we take advantage of GIN (GIN-E in case edge features are also present) architecture [XHLJ19, LL22]. Specially, we will use GIN (GIN-E) AGGREGATE and UPDATE functions. For GIN, AGGREGATE and UPDATE functions are defined as:

$$\text{AGGREGATE}_{\theta_1^{(l)}}\left(\left\{\!\!\left\{\left(x_w^{(l-1)}, e_{(w,v)}^{(l-1)}\right)\right\}\!\!\right\}_{w \in N(v)}\right) = \sum_{w \in N(v)} x_w^{(l-1)}$$

$$\text{UPDATE}_{\theta_2^{(l)}}\left(x_v^{(l-1)}, m_v^{(l)}\right) = MLP_{\theta_2^{(l)}}\left((1+\epsilon) \cdot x_v^{(l-1)} + m_v^{(l)}\right)$$

GIN-E also takes advantage of the edge embeddings:

$$\text{AGGREGATE}_{\theta_1^{(l)}}\left(\left\{\!\!\left\{\left(x_w^{(l-1)}, e_{(w,v)}^{(l-1)}\right)\right\}\!\!\right\}_{w \in N(v)}\right) = \sum_{w \in N(v)} x_w^{(l-1)} + e_{(w,v)}^{(l-1)}$$

$$\text{UPDATE}_{\theta_2^{(l)}}\left(x_v^{(l-1)}, m_v^{(l)}\right) = MLP_{\theta_2^{(l)}}\left((1+\epsilon) \cdot x_v^{(l-1)} + m_v^{(l)}\right)$$

Note that MLP denotes Multilayer perceptron and ϵ is user-specified or trainable parameter.

2.4.3 Expressive power

The expressive power of GNNs refers to their ability to distinguish two non-isomorphic graphs. The greater the expressive power of a model, the better its ability to capture and represent complex relationships and patterns in graph-structured data. Formally, we can define expressive power as follows:

Consider the space $\mathcal{K}$ of all graphs. Then we define a space $\mathcal{K}(f)$ as a subset of quotient space $\mathcal{K} \,/\, \cong$ consisting of all graphs that are uniquely mapped by model f to distinct objects, so for any two graphs $T, G \in \mathcal{K}(F) : f(G) \neq f(T)$. Then the expressive power of model f is given as a size of $\mathcal{K}(f)$ [LL22]. We define model f_1 as being more expressive than model f_2 if $\mathcal{K}(f_2) \subset \mathcal{K}(f_1)$.

The expressive power of GNNs is closely connected to the graph isomorphism problem. Determining if the two graphs are isomorphic is a difficult problem due to the absence of any polynomial-time algorithm discovered so far [GJ79]. The quasipolynomial algorithm was proposed in [Bab16]. However, there are other, less sophisticated algorithms for graph isomorphism that can successfully distinguish a large set of graphs, but they cannot guarantee to differentiate all non-isomorphic graphs.

A family of such algorithms that are widely used due to their efficiency and their ability to distinguish a broad class of graphs is the Weisfeiler-Leman (WL) tests [WL68]. Specially, its 1-dimensional version (1-WL) works by iteratively refining the labels assigned to the nodes of the graphs being compared. In the initial step, a unique label is assigned to each node. In the subsequent iterations, the labels are updated using an injective function that depends on the previous label of a node and the labels of its neighboring nodes. This process continues until stable labeling is reached. If the final labelings of the two graphs are the same, they are considered to be isomorphic. Otherwise, they are regarded as non-isomorphic. Algorithm 2 shows the pseudocode of the 1-WL test.

Algorithm 2 1-WL test. HASH is an injective mapping, so different tuples are mapped to different labels

Require: Graphs G_1 and G_2
  $l_v^{(1,0)}$: Initial unique node labels $\forall v \in V(G_1)$
  $l_v^{(2,0)}$: Initial unique node labels $\forall v \in V(G_2)$
  for i = 1, 2, ..., until stable labeling **do**
    $l_v^{(1,i)} \leftarrow \text{HASH}\Big(l_v^{(1,i-1)}, \{\!\{ l_w^{(1,i-1)} \mid w \in N(v) \}\!\}\Big) \;\forall v \in V(G_1)$
    $l_v^{(2,i)} \leftarrow \text{HASH}\Big(l_v^{(2,i-1)}, \{\!\{ l_w^{(2,i-1)} \mid w \in N(v) \}\!\}\Big) \;\forall v \in V(G_2)$
    if $\big\{ l_v^{(1,i)} \mid v \in V(G_1) \big\} \neq \big\{ l_v^{(2,i)} \mid v \in V(G_2) \big\}$ **then**
      return $G_1 \ncong G_2$
    end if
  end for
  return $G_1 \cong G_2$

When the 1-WL test identifies two graphs as non-isomorphic, we can confidently

conclude that they are indeed non-isomorphic. However, there exist cases where the 1-WL test suggests that two non-isomorphic graphs are isomorphic. In such cases, the 1-WL test fails to discriminate between those two graphs.

When comparing the 1-WL test to a standard GNN architecture, it can be seen that the AGGREGATE and UPDATE functions in the GNN can be interpreted as an implementation of a HASH function employed within the 1-WL test. Likewise, the READOUT operation can be regarded as a consolidation of the node representations.

This analogy allows us to quantify the expressive power of the standard GNN by means of the 1-WL test. Notably, it has been proven that a standard GNN model can be at most as expressive as 1-WL. In other words, if a GNN model maps two graphs to different graph embeddings, then 1-WL also decides that those two graphs are not isomorphic [XHLJ19, MRF+19]. Furthermore, it has been proved that standard GNN can reach the expressive power of 1-WL if the following two conditions hold:

- The AGGREGATE and UPDATE functions construct an injective mapping.
- The READOUT function is also injective.

2.5 Hierarchical Inter-Message Passing

Hierarchical Inter-Message Passing scheme (HIMP) [FYW20] is a novel architecture for GNNs. HIMP was developed to address some known limitations of standard GNNs, particularly their inability to differentiate between certain molecules, as illustrated in Figure 2.11. These drawbacks primarily arise from the inability of GNNs to detect cycles within the graph structure since they are unable to maintain information about which node has contributed what to the aggregated information [Lou20, HTP+18]. Therefore HIMP incorporates a Feature Tree capable of representing cycles along the original molecular graph. This increases the network's expressive power as it allows reasoning about the hierarchy, e.g., rings, in molecules, and ultimately overcoming the shortcomings of GNNs. Consequently, HIMP can achieve state-of-the-art performance across various datasets.

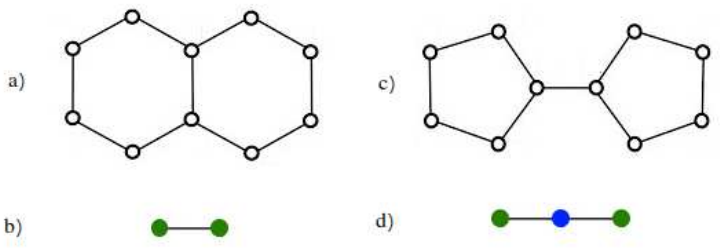

Figure 2.11: Molecular graphs of Decalin a) and Bicyclopentyl c), and their corresponding Feature Trees b), d), respectively. A standard GNN is not able to distinguish those two molecules based on their molecular graphs because they are 1-WL equivalent. However, the FTs for those two molecules can clearly be distinguished. So if we take both representations into account during training, we can learn different graph embedding for those two molecules.

2.5.1 Architecture

Rather than relying on a single GNN operating solely on the molecular graph, HIMP leverages two separate GNN models. One GNN model operates on the molecular graph, while the other model operates on its corresponding Feature Tree. Each of these two GNN models performs intra-message passing, i.e., the exchange of information with neighboring nodes within their respective representations as the standard GNNs. Additionally, this scheme is enhanced by incorporating inter-message passing. Within each layer, the information is exchanged between the two graph representations as follows:

Let G be the molecular graph and T be its corresponding FT. Recall that each node of FT corresponds to a set of nodes of the molecular graph. Let $S \in \{0,1\}^{|V(G)|\times|V(T)|}$ be the mapping matrix that captures the assignment of nodes of the molecular graph $v \in V(G)$ to nodes of the FT $C \in V(T)$ (we refer to nodes of FT as clusters). $S_{v,i} = 1$ if and only if $v \in C_i$.

Now let $\mathbf{X}^{(l)} \in \mathbb{R}^{|V(G)|\times d_x}$ and $\mathbf{T}^{(l)} \in \mathbb{R}^{|V(T)|\times d_x}$ be matrices that store the embeddings of nodes of G and T in layer l, respectively. Then the inter-message passing step changes the embedding matrices $\mathbf{X}$ and $\mathbf{T}$ as follows:

$$\mathbf{X}^{(l)} \leftarrow \mathbf{X}^{(l)} + \sigma\left(S\mathbf{T}^{(l)}W_1^{(l)}\right)$$

$$\mathbf{T}^{(l)} \leftarrow \mathbf{T}^{(l)} + \sigma\left(S^T\mathbf{X}^{(l+1)}W_2^{(l)}\right),$$

where σ denotes a non-linearity and matrices $W_1^{(l)}, W_2^{(l)} \in \mathbb{R}^{d_x\times d_x}$ are trainable parameters within layer l.

Of particular importance is that each node of G is aware of its cluster assignment. Furthermore, it possesses knowledge of which other nodes are part of the same cluster.

This results in an enhanced expressivity of the model, as shown in Figure 2.11 and the enhanced performance of the model.

The READOUT is then defined as follows:

$$\frac{1}{|V(G)|}\sum_{i=1}^{|V(G)|} x_i^{(L)} \;\Big\|\; \frac{1}{|V(T)|}\sum_{i=1}^{|V(T)|} t_i^{(L)},$$

where $||$ denotes the concatenation operator, $x_i^{(L)} \in \mathbb{R}^{d_x}$ and $t_i^{(L)} \in \mathbb{R}^{d_x}$ hold for the final embedding of the i-th node of graph G and T, respectively. It is worth noting that the original paper specifies the READOUT function as:

$$\sum_{i=1}^{|V(G)|} x_i^{(L)} \;\Big\|\; \sum_{i=1}^{|V(T)|} t_i^{(L)}$$

However, in the provided implementation, they utilize the mean READOUT function. In order to ensure comparability with the results presented in the paper, we will consider the mean READOUT function as our foundation for further extensions.

The high-level overview of how HIMP works is depicted in Figure 2.12.

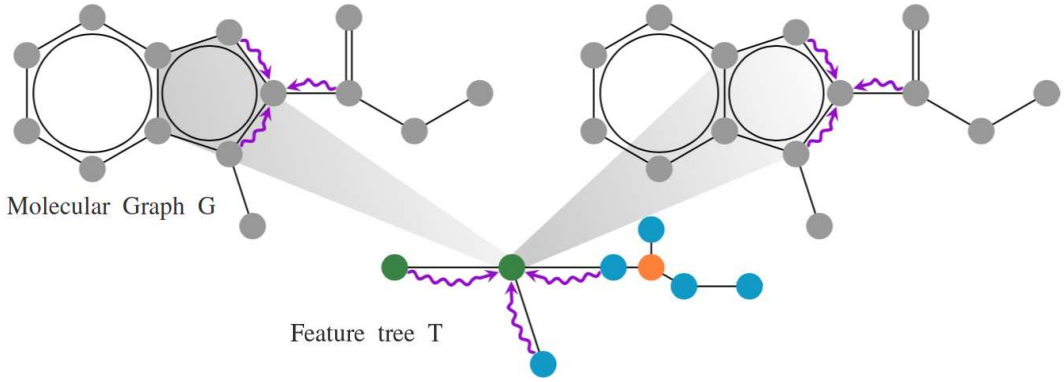

Figure 2.12: An overview of Hierarchical Inter-Message Passing scheme. Two GNNs are trained on distinct graphs G and T. After each intra-message passing step, the two graph representations mutually exchange the information (inter-message passing step) [FYW20].

3 Extending HIMP

In this chapter, we present our extensions to HIMP. Even though HIMP is able to reach state-of-the-art performance and overcome some known limitations of GNNs, further improvements still can be made. Therefore we extend the existing HIMP, potentially making it more expressive and enhancing its overall performance.

3.1 ErG for HIMP

Firstly, we enable HIMP to take advantage of Extended Reduced Graph that can effectively capture the pharmacophoric features of the molecule as well as the topological relationships between atoms within the molecule. This allows us to encode more different aspects of the molecule than if we only use FT. Those aspects may be crucial for predicting some properties, so the usage of ErG may greatly enhance the performance of the model.

It is important to note that usage of ErG along the molecular graph also increases the expressive power of the model. For example, a model that uses ErG together with the molecular graph can distinguish Decalin and Bicyclopentyl, in contrast to standard GNN - see Figure 3.1.

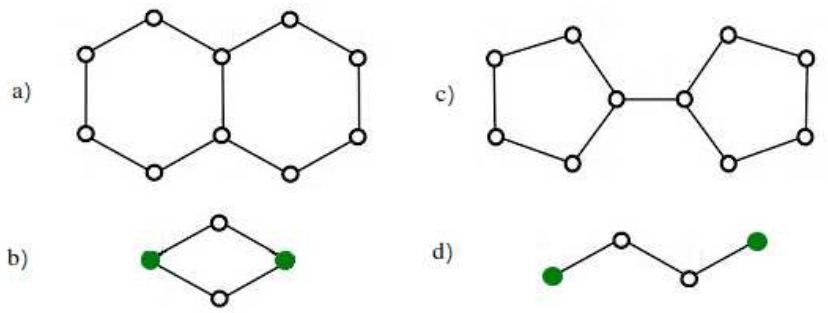

Figure 3.1: Molecular graphs of Decalin a) and Bicyclopentyl c), and their corresponding Extended Reduced Graphs b), d), respectively. A standard GNN can't distinguish those two molecules based on their molecular graphs. However, the ErGs for those two molecules can clearly be distinguished by the 1-WL test.

Each node of ErG is assigned one of the following seven features: no-feature, donor, acceptor, positive charge, negative charge, hydrophobic group, and aromatic ring system.

Similarly to the usage of FT within HIMP, each node within the molecular graph that belongs to a hydrophobic group or aromatic ring system, is aware of all other atoms present within the same group or system. Naturally, each atom maintains knowledge of its own assigned feature.

The usage of ErG along the molecular graph follows a similar approach to the utilization of FT in the original HIMP framework introduced in Subsection 2.5, particularly with regards to the architecture.

3.2 Arbitrary number of graphs

Since each graph representation captures the different aspects of molecules, it may be highly beneficial to use all graph representations. By utilizing multiple graph representations, we can achieve a more comprehensive model, offering not only enhanced expressiveness but also increased predictive power.

In this section, we present the extended HIMP architecture that allows us to operate on an arbitrary number of graphs. Within each layer, the molecular graph representation exchanges information with every other graph representation. Formally:

Let $\mathbf{X}^{(l)} \in \mathbb{R}^{|V(G)| \times d_x}$ be the embedding matrix of the molecular graph G in layer l, and $\mathbf{T}_1^{(l)}, \mathbf{T}_2^{(l)}, ..., \mathbf{T}_n^{(l)}$ be the embedding matrices of the corresponding reduced graphs $T_1, T_2, ..., T_n$ in layer l (it is worth noting that these may be arbitrary graph representations, not necessarily reduced graphs). Note that, $\forall i = 1, 2, ..., n : \mathbf{T}_i^{(l)} \in \mathbb{R}^{|V(T_i)| \times d_x}$. The inter-message passing step changes the intermediate embedding matrices as follows:

$$\mathbf{X}^{(l)} \leftarrow \mathbf{X}^{(l)} + \sum_{i=1}^{n} \sigma\left(S_i \mathbf{T}_i^{(l)} W_{i,1}^{(l)} \right)$$

For $i = 1, 2, ..., n$:

$$\mathbf{T}_i^{(l)} \leftarrow \mathbf{T}_i^{(l)} + \sigma\left(S_i^T \mathbf{X}^{(l+1)} W_{i,2}^{(l)} \right),$$

where $W_{i,1}^{(l)}, W_{i,2}^{(l)} \in \mathbb{R}^{d_x \times d_x}$ are trainable parameters within layer l, $S_i \in \{0,1\}^{|V(G)| \times |V(T_i)|}$ are the mapping matrices that stores the assignment of nodes of the molecular graph to nodes of the i-th reduced graph, and σ denotes nonlinearity. It is important to note that for our purposes, $\mathbf{X}$ must indeed be the embedding matrix of G since we require a mapping between $\mathbf{X}$ and all $\mathbf{T}_i$. As reduced graphs are constructed from G, this mapping will always be present.

Finally, we define our READOUT function as the concatenation of mean READOUTs of individual graph representations, each weighted by a trainable matrix:

$$h_G = W_0 \frac{1}{|V(G)|} \sum_{i=1}^{|V(G)|} x_i^{(L)} \Big|\Big| W_1 \frac{1}{|V(T_1)|} \sum_{i=1}^{|V(T_1)|} t_{1,i}^{(L)} \Big|\Big| ... \Big|\Big| W_n \frac{1}{|V(T_n)|} \sum_{i=1}^{|V(T_n)|} t_{n,i}^{(L)},$$

with $||$ denoting the concatenation operator, $W_0, W_1, ..., W_n \in \mathbb{R}^{d_x \times d_x}$ stand for trainable matrices, $x_i^{(L)} \in \mathbb{R}^{d_x}$ holds for the final embedding of the i-th node of graph G after

L layers, similarly $t_{j,i}^{(L)} \in \mathbb{R}^{d_x}$ denotes the final embedding of the i-th node of the j-th reduced graph.

4 Experiments

As outlined in Section 2.5, HIMP is a powerful Graph Neural Network architecture that has achieved state-of-the-art performance across multiple datasets. Building upon the foundation of HIMP, in Chapter 3, we introduced a novel architecture that incorporates Extended Reduced Graph representation and is able to operate on an arbitrary number of graph representations.

This chapter aims to present a setup for a comprehensive set of experiments to evaluate the performance of our novel architecture on well-established benchmark datasets for molecular prediction. Subsequently, we compare the achieved results with those of various state-of-the-art GNN techniques.

All models were trained using the training set. The hyperparameters were fine-tuned using the validation set. Employing a shared set of hyperparameters for all methods in this work ensured a consistent comparison between different approaches. This approach not only reduced the complexity of managing multiple hyperparameter configurations but also facilitated clear and straightforward comparisons across various methods. Moreover, it simplified the experimental setup and made it more manageable, especially when dealing with resource constraints like limited computational resources or time.

Ultimately, the performance evaluation of the models was conducted using the independent test set.

4.1 Models

In this section, we present a comprehensive collection of models that form the foundation for our experiments, all based on our proposed architecture. These models are then compared with state-of-the-art GNN techniques, including the Graph Convolutional Network (GCN) [KW17] alongside its variant that incorporates edge features as well - GCN-E, the Graph Isomorphism Network (GIN) [LL22] with its variant that uses also edge features - GIN-E [HLG$^+$20], the Residual Gated Graph Convolution Network (GatedGCN) also with variant that uses edge features - GatedGCN-E [BL18], the Graph Sample and Aggregate method (GraphSAGE) [HYL17], the Graph Attention Network (GAT) [VCC$^+$18], the Monti Network (MoNet) [MBM$^+$17], the Neural Graph Fingerprint (NGF) mehtod [DMAI$^+$15], the Relation Pooling Neural Graph Fingerprint (RP-NGF) [MSRR19], and the Reduced-Graph Message-Passing Neural Network (RG-MPNN) [KZL$^+$22].

Our collection consists of nine models. Firstly, we include models that utilize a single graph representation, falling under the category of standard GNN models. We benchmark three such models, one that solely employs the raw molecular graph - RAW-Only, one that uses only FT representation - FT-Only, and one that utilizes only ErG - ErG-Only.

Models	Molecular graph	FT	ErG	Inter-message Passing
RAW-Only	✓			
FT-Only		✓		
ErG-Only			✓	
RAW||FT	✓	✓		
RAW||ERG	✓		✓	
RAW||FT||ERG	✓	✓	✓	
RAW+FT	✓	✓		✓
RAW+ERG	✓		✓	✓
RAW+FT+ERG	✓	✓	✓	✓

Table 4.1: Overview of proposed HIMP based GNN models

These models allow us to investigate the individual predictive capabilities of each graph representation for molecular properties.

Next, we incorporate models based on our Extended HIMP scheme. Specifically, we utilize the standard HIMP model that uses FT with the raw molecular graph - RAW+FT, then the model that incorporates ErG alongside the raw molecular graph - RAW+ERG, and model that take advantage of both reduced graph representations along the raw molecular graph- RAW+FT+ERG. Our objective here is to investigate the potential enhancement of the model through the inclusion of ErG, as well as the benefits that may arise from utilizing multiple graph representations.

In addition, we evaluate alternative versions of the three models mentioned in the previous paragraph, where inter-message passing is not employed. Those models only train multiple GNNs in parallel and subsequently output concatenation of the individual GNN outputs. These variants include RAW||FT, RAW||ERG, and RAW||FT||ERG. By comparing the versions that utilize inter-message passing with their counterparts that do not, we can effectively study and evaluate the benefits obtained from this approach.

All our proposed models were implemented using Pytorch [PGM+19], Pytorch geometric [FL19], and RDKit software [Lan16]. Our newly proposed models are summarized in Table 4.1.

4.1.1 Architecture

Even though the presented models differ in some aspects, they share some similarities in their architecture:

- All the models were trained using Adam optimizer with learning rate $\alpha = 0.0001$, decay rates $\beta_1 = 0.9, \beta_2 = 0.999$, and $\epsilon_A = 10^{-8}$. Only for the ogbg-molpcba, we

used $\alpha = 0.001$, and for the ZINC dataset, we initially set the learning rate to $\alpha = 0.001$. Then, if the loss function does not show any improvement for 10 epochs, we take the step of halving the learning rate. The learning rate was allowed to decrease up to a value of 2^{-5}.

- The nonlinearity σ within HIMP is implemented by the GIN operator (GIN-E if edge features are present - this is the case only for the raw molecular graph)
  - The Multilayer perceptor within GIN (Graph Isomorphism Network) consists of the input and output layer of size d_x and one hidden layer of size $2d_x$, where d_x denotes a so-called **hidden size**. Nonlinearities ϕ and ψ are formed by first applying batch normalization ($\epsilon_{bn} = 10^{-5}$) followed by ReLU.
  - The ϵ parameter of GIN (GIN-E)is set to be trainable
- The READOUT of a model is the model is then transformed to prediction using the following function:

$$\hat{y} = W_o \left(\mathrm{ReLU}\big(\underbrace{[I_{d_x}, ..., I_{d_x}]}_{r \text{ times}} h_G\big)\right),$$

  with $[\cdot, ..., \cdot]$ denoting the concatenation, I_{d_x} representing the identity matrix of size d_x, r standing for the number of graph representations, and $W_o \in \mathbb{R}^{o \times d_x}$ being the trainable matrix (o is a number of output variables). So the input to the ReLU function is a vector whose i-th element is obtained as a sum of i-th elements of final graph embeddings of all input graphs.
- All other hyperparameters (number of epochs, number of hidden layers, hidden size, batch size, dropout ratio) were fine-tuned using the validation datasets. Since the values of these hyperparameters vary for each benchmark dataset, they are discussed in the following section dedicated to the datasets.

4.2 Datasets

In this section, we present the benchmarking datasets.

4.2.1 MoleculeNet Datasets

The MoleculeNet dataset collection covers multiple aspects of molecular research, including molecular graph regression, binary classification, multi-task classification, and property optimization [WRF+18]. It covers a wide variety of datasets obtained from different sources, ensuring a comprehensive representation of chemical space and diverse molecular properties.

We benchmarked our model on three datasets from this collection. For all three datasets, we used 80%/10%/10% random split.

To measure the performance of the models on those datasets, we used the AUC-ROC metric.

Tox21 This dataset focused on predicting the toxicological properties of chemical compounds, providing valuable insights for drug safety assessment. To assess the performance of our models on this dataset, we utilize the following hyperparameters:

Number of epochs	150
Number of hidden GNN layers	3
Hidden size	192
Batch size	128
Dropout ratio	0.25

MUV MUV is designed to evaluate the performance of machine learning models in virtual screening tasks, specifically in identifying active compounds against particular targets. For this dataset, we used the following hyperparameters:

Number of epochs	100
Number of hidden layers	2
Hidden size	64
Batch size	96
Dropout ratio	0.5

HIV A dataset specifically designed for predicting the activity of compounds against the HIV virus, aiding in the development of antiretroviral drugs. To evaluate the performance of our models on this dataset, we employ the following set of hyperparameters:

Number of epochs	60
Number of hidden layers	2
Hidden size	256
Batch size	128
Dropout ratio	0.5

4.2.2 ZINC Dataset

The ZINC FULL dataset [GBWD+18] comprises approximately 250,000 molecules randomly sampled from the extensive ZINC database [ISM+12]. This dataset consists of molecules with up to 38 heavy atoms. The objective is to regress a specific molecular property known as constrained solubility.

For the purpose of our study, we will focus on a specific subset of the ZINC FULL dataset. This subset, consisting of 12,000 molecules, was defined and outlined in [DJL+23]. The dataset is divided into three predefined sets: the training set consisting of 10,000 molecules, the validation set containing 1,000 samples, and the test set comprising another 1,000 molecules.

To assess the performance of models on this dataset, we employed the Mean absolute error metric.

We utilized the following set of hyperparameters:

Number of epochs	400
Number of hidden layers	5
Hidden size	320
Batch size	256
Dropout ratio	0.1

4.2.3 Open Graph Benchmark datasets

The Open Graph Benchmark [HFZ+20] is an extensive benchmarking framework designed specifically for graph machine learning tasks. It provides a diverse range of benchmark datasets, along with predefined evaluation metrics and predefined train-validation-test splits. This comprehensive framework ensures that GNN models can be consistently and rigorously evaluated, enabling accurate assessment of their performance.

We assessed the performance of the models on two selected benchmark datasets from the Open Graph Benchmark.

ogbg-molhiv dataset

This dataset is identical to the HIV dataset from the MoleculeNet dataset collection. The key distinction lies in the different approach for splitting the data into training, validation, and test sets called scaffold splitting procedure [WRF+18].

The goal of scaffold splitting is to ensure that molecules with similar core structures are not distributed across different sets, as this could introduce bias or result in overly optimistic performance estimates. Scaffold splitting helps to mimic real-world scenarios where a model needs to generalize to new molecules with different scaffolds. Preserving scaffold diversity across sets provides a more realistic and reliable evaluation of the model's performance on unseen data.

We employed the AUC-ROC as a performance measure and utilized a specific set of hyperparameters, which are as follows:

Number of epochs	50
Number of hidden layers	3
Hidden size	128
Batch size	256
Dropout ratio	0.5

ogbg-molpcba

This dataset is used for developing and evaluating models that can predict the biological activity of molecules based on their chemical features. We also utilized the scaffold splitting procedure. Since the dataset is highly imbalanced, we use AUC-PRC as a performance measure.

The set of hyperparameters used for training our models on this dataset:

Number of epochs	150
Number of hidden layers	3
Hidden size	256
Batch size	512
Dropout ratio	0.25

5 Results

In this chapter, we present and discuss the results of our study, focusing on the performance comparison between our newly developed models and state-of-the-art approaches. We evaluate the performance across various datasets to assess the effectiveness of our models in addressing the research objectives. Through a thorough analysis and discussion of the results, we aim to provide insights into the strengths and limitations of our models, as well as their potential contributions to the field.

All the results were obtained as a mean of five runs of the algorithms. Throughout this chapter, we have multiplied the values of AUC-ROC and AUC-PRC by 100 for the purpose of enhancing readability and comprehension.

The displayed results in the tables are differentiated by a grey shading, with our experiment results being shaded, while results extracted from other papers remain unshaded.

5.1 MoleculeNet datasets

The performance comparison of various models on the three datasets from the MoleculeNet collection is presented in Table 5.1. Recall that the higher AUC-ROC indicates the better performance of the model. The results of NGF and RP-NGF are obtained from [MSRR19].

Notably, most of our proposed models exhibit superior performance compared to the NGF and RP-NGF across all datasets. This is particularly promising considering that RP-NGF has higher expressive power than all our models, as it can distinguish any two graphs. However, despite this advantage, it demonstrates poorer overall generalization.

However, ErG-Only and FT-Only models perform worse than all three state-of-the-art methods, which is expected since they rely on reduced graph representations that may lack important information from the original graph.

Interestingly, ErG-Only consistently outperforms FT-Only on all three datasets, possibly due to the fact that ErG preserves more information by reducing the original graph to a lesser extent than FT. For almost every molecule, the corresponding ErG consists of more nodes and edges than the corresponding FT. It is worth mentioning that both models demonstrate high stability with low standard deviations.

Arguably, the most important information is that RAW+FT+ERG model achieves the best results on HIV and MUV datasets, surpassing other models significantly in the case of the MUV dataset. These findings support our hypothesis presented in Chapter 3 that utilizing more than two graph representations within the HIMP scheme may enhance the performance because each representation may capture different aspects of the molecule. RAW+FT+ERG was outperformed only by RG-MPNN on Tox21 dataset.

Furthermore, the benefit of inter-message passing is evident as models incorporating this feature generally outperform their counterparts without inter-message passing. The exception is the HIV dataset, where RAW||FT achieves slightly better results than RAW+FT but with a higher standard deviation. Additionally, models with inter-message passing exhibit greater stability compared to those without inter-message passing.

Notably, the combination of ErG and the original graph demonstrates comparable performance compared to the combination of FT and the original graph, especially when inter-message passing is employed. RAW+ERG outperforms RAW+FT on two datasets, with only marginally worse results on the MUV dataset.

In summary, our experimental results highlight the effectiveness of our proposed models, particularly the RAW+FT+ERG model, which consistently achieves very good performance. Those experimental results strongly support the idea that the utilization of inter-message passing and the inclusion of different graph representations within the HIMP framework may enhance model performance.

Methods	**ROC-AUC**		
	HIV	MUV	Tox21
NGF	81.20±1.40	79.80±2.50	79.40±1.00
RP-NGF	83.20±1.30	79.40±0.50	79.90±0.60
RG-MPNN	82.40±1.90	81.90±1.10	**87.30**±0.80
RAW-Only (GIN-E)	83.88±0.63	80.77±0.84	84.31±0.35
ErG-Only	65.27±0.05	69.07±0.03	70.01±0.02
FT-Only	63.23±0.11	63.56±0.21	63.14±0.13
RAW||FT	85.07±0.67	80.98±0.55	84.57±0.44
RAW||ERG	84.98±0.70	81.41±0.51	84.51±0.41
RAW||FT||ERG	85.50±0.51	81.64±0.35	84.91±0.29
RAW+FT	84.92±0.35	81.91±0.42	85.07±0.05
RAW+ERG	85.36±0.12	81.82±0.10	85.27±0.09
RAW+FT+ERG	**85.51**±0.21	**84.22**±0.17	85.46±0.32

Table 5.1: Performance comparison of GNN models on three datasets from the MoleculeNet collection. The first section shows the performance of selected state-of-the-art models. The second shows the performance of standard GNNs operating on a single graph representation, the third section shows the performance of models that train multiple GNNs and then concatenate outputs, and finally, the fourth section shows the performance of HIMP-based models.

5.2 ZINC dataset

Table 5.2 displays the performance comparison of different models on the ZINC dataset. Note that this is a graph regression task. The evaluation metric used in this comparison is MAE, where a smaller value indicates better performance of the model. We evaluate

our models against all the baseline models described in [DJL+23], and we also include the results of those baseline methods reported in the same paper.

Those results show that our models can also outperform state-of-the-art models on graph regression tasks. All of them (excluding ErG-Only and FT-Only) achieve much better results than all state-of-the-art methods. Especially the method with inter-message passing outperformed them by a lot.

The results highlight that concatenation results from multiple GNNs may lead to worse performance. It can be seen that RAW-Only outperforms all three methods that train multiple GNNs on different graph representations and concatenate the outputs - RAW||FT, RAW||ERG, RAW||FT||ERG. However, we can see that all methods that utilize the inter-message passing significantly outperform their alternatives that use concatenation and even RAW-Only. This shows that if we train multiple GNNs independently and just concatenate their outputs, they may fail to complement each other which may result in worse performance, and adding more graph representations may even further decrease the performance - it can be seen that RAW||FT||ERG is outperformed by both RAW||ERG and RAW||FT.

Nevertheless, if we exchange information via inter-message passing, it seems to greatly enhance the performance and usage of more graph representations may be even more beneficial - it can be seen that HIMP-based model achieves the best results and RAW+FT+ERG outperforms both RAW+ERG and RAW+FT.

5.3 Open Graph Benchmark datasets

Table 5.3 presents a comprehensive comparison of the performance of various models. In line with the experimental protocol outlined in [HFZ+20], we assessed the models' performance using the AUC-ROC and AUC-PRC metrics. It is important to note that higher values indicate superior model performance for both metrics. Our proposed model was benchmarked against GCN-E and GatedGCN-E, with their results obtained from HIMP.

Similarly to the results obtained on other datasets, we can observe that all our models (except ErG-Only and FT-Only) achieved better performance than the state-of-the-art models.

Models solely relying on a reduced graph representation (ErG-Only, FT-Only) exhibited poorer performance compared to other models. This phenomenon was consistently observed across all datasets and aligns with our earlier discussions. Notably, on the ogbg-molpcba dataset, these two models yielded significantly inferior results compared to all other methods.

Interestingly, the RAW-Only model demonstrated impressive results for both datasets, particularly on ogbg-molpcba, where it even outperformed all models except RAW+FT+ERG. This finding suggests that RAW-Only can be highly beneficial for imbalanced datasets.

Further analysis reveals that the usage of ErG is more or less equivalent to the usage of FT for both datasets. Methods incorporating ErG along the molecular graph (RAW||ERG,

Methods	MAE ZINC
GatedGCN	0.435±0.011,
GIN	0.408±0.008,
GraphSAGE	0.398±0.002,
GAT	0.384±0.007,
GCN	0.367±0.011,
MoNet	0.292±0.006,
GatedGCN-E	0.282±0.015,
RAW-Only (GIN-E)	0.223±0.003,
ErG-Only	1.064±0.001
FT-Only	1.140±0.001
RAW||FT	0.231±0.010
RAW||ERG	0.227±0.005
RAW||FT||ERG	0.2333±0.009
RAW+FT	0.194±0.006
RAW+ERG	0.193±0.002
RAW+FT+ERG	**0.187**±0.003

Table 5.2: A performance comparison of GNN models on ZINC dataset. The first section showcases the performance of selected state-of-the-art models. The second section displays the performance of standard GNNs operating on a single graph representation. The third section shows the performance of models that train multiple GNNs and then concatenate their outputs. Finally, the fourth section highlights the performance of HIMP-based models.

RAW+ERG) exhibited similar results to models leveraging FT along the molecular graph (RAW||FT, RAW+FT).

Once again, the most noteworthy result is that RAW+FT+ERG outperformed all other proposed models, as well as the state-of-the-art methods. This finding underscores the model's potential to achieve not only the best results on other presented datasets but also exceptional performance on imbalanced datasets.

Methods	AUC-ROC ogbg-molhiv	AUC-PRC ogbg-molpcba
GCN-E	76.07±0.97,	19.83±0.16,
GatedGCN-E	77.65±0.50,	20.77±0.27,
RAW-Only (GIN-E)	76.87±0.28,	26.52±0.35,
ErG-Only	67.60±0.40	5.58±0.04,
FT-Only	62.03±0.58	3.51±0.02,
RAW||FT	78.17±0.91	25.86±0.42,
RAW||ERG	77.26±0.62	26.60±0.31,
RAW||FT||ERG	77.93±1.81	26.35±0.24,
RAW+FT	77.47±0.86	26.13±0.30,
RAW+ERG	77.65±0.61	25.36±0.28,
RAW+FT+ERG	**78.81**±0.80	**26.70**±0.86,

Table 5.3: A comparison of results obtained by GNN models on two datasets from Open Graph Benchmark collection. The first section displays the performance of selected state-of-the-art models. Subsequently, the second section showcases the performance of standard GNNs operating on a single graph representation. The following section presents the performance results of models that employ the training of multiple GNNs followed by the concatenation of their outputs. Lastly, the fourth section presents the performance outcomes of models based on the HIMP approach.

6 Conclusion

In conclusion, this thesis has introduced a novel graph learning technique based on the Hierarchical Inter-Message Passing scheme, incorporating various reduced graph representations, including the Extended Reduced Graph and Feature Tree approach, alongside the original molecular graph. The main goal of this thesis was to develop enhanced graph learning techniques that surpass state-of-the-art methods in the field of drug discovery.

Chapter 3 presented the extensions made to HIMP, specifically the utilization of ErG and the ability to incorporate an arbitrary number of graphs within the HIMP framework. Building upon this framework, Chapter 4 proposed multiple GNN models to investigate the benefits of using ErG within HIMP, the potential performance improvements of utilizing more than two graphs, and the impact of inter-message passing.

Based on the results obtained in Section 5, it can be concluded that incorporating the ErG scheme alongside the molecular graph significantly enhances the model's performance. The proposed methods (RAW||ERG, RAW+ERG) outperformed the model using only the molecular graph (RAW-Only) on all presented datasets, except for the ZINC dataset. Additionally, these methods achieved comparable results to those utilizing the FT representation (RAW||FT, RAW+FT).

Furthermore, the inclusion of inter-message passing proved to be consistently beneficial, as the methods employing inter-message passing outperformed their counterparts without inter-message passing on almost all datasets.

Ultimately, the findings demonstrate that by combining both extensions to HIMP and creating a model that incorporates FT, ErG, and the original graph, along with inter-message passing (RAW+FT+ERG), we can outperform all other proposed methods, as well as state-of-the-art methods, across the vast majority of benchmark datasets presented. This provides us with a powerful model for accurately predicting molecular properties.

6.1 Future work

There are several possibilities for future research and exploration based on the findings of this thesis. Two potential directions for further investigation are discussed below.

Firstly, it would be interesting to investigate whether the combined usage of ErG, FT, and the molecular graph can further increase the expressive power of the model compared to using only FT and the molecular graph. While in [FYW20] was demonstrated that the incorporation of FT alongside the molecular graph increases the expressive power of a model, combining ErG with FT and the molecular graph may provide an even more comprehensive representation of molecular structures.

Secondly, the Feature Tree allows for the description of molecules at various levels of resolution. By incorporating feature trees with different levels of resolution, the model could capture and leverage information at varying levels of granularity, potentially leading to more accurate predictions of molecular properties. Exploring the impact of utilizing FT with various levels of resolution would provide valuable insights into the optimal level of detail needed to achieve the best performance in molecular property prediction.

Bibliography

[Agg18] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing, 2018.

[Bab16] László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, page 684–697, New York, NY, USA, 2016. Association for Computing Machinery.

[Bar14] Ronald Bartzatt. Determination of dermal permeability coefficient (kp) by utilizing multiple descriptors in artificial neural network analysis and multiple regression analysis. *Journal of Scientific Research and Reports*, 3:2884–2899, 01 2014.

[BG11] Kristian Birchall and Valerie J. Gillet. *Reduced Graphs and Their Applications in Chemoinformatics*, pages 197–212. Humana Press, Totowa, NJ, 2011.

[BL18] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2018.

[BO04] Albert-Laszlo Barabasi and Zoltan Oltvai. Network biology: Understanding the cell's functional organization. *Nature reviews. Genetics*, 5:101–13, 03 2004.

[CAEHP+22] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *Journal of Machine Learning Research*, 23(89):1–64, 2022.

[CCVB20] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[con20] Wikimedia Commons contributors. Artificial neural network. Online: `https://commons.wikimedia.org/w/index.php?title=Artificial_neural_network&oldid=428071248`, 21 June 2020.

[Con23a] Wikipedia Contributors. Graph (discrete mathematics). Online: `https://en.wikipedia.org/w/index.php?title=Graph_(discrete_mathematics)&oldid=1149629694`, 13 April 2023.

[Con23b] Wikipedia Contributors. Caffeine. Online: `https://en.wikipedia.org/w/index.php?title=Caffeine&oldid=1152789240`, 2 May 2023.

[Con23c] Wikipedia Contributors. Tree (graph theory). Online: `https://en.wikipedia.org/w/index.php?title=Tree_(graph_theory)&oldid=1146024536`, 30 July 2023.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[DDJ+22] Suresh Dara, Swetha Dhamercherla, Surender Singh Jadav, CH Madhu Babu, and Mohamed Jawed Ahsan. Machine learning in drug discovery: A review. *Artificial Intelligence Review*, 55(3):1947–1999, Mar 2022.

[DJL+23] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

[DKZ+17] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6351–6361, Red Hook, NY, USA, 2017. Curran Associates Inc.

[DMAI+15] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 2224–2232, Cambridge, MA, USA, 2015. MIT Press.

[FL19] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[FLM+20] Matthias Fey, Jan Eric Lenssen, Christopher Morris, Jonathan Masci, and Nils M. Kriege. Deep graph matching consensus. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[FYW20] Matthias Fey, Jan-Gin Yuen, and Frank Weichert. Hierarchical inter-message passing for learning on molecular graphs. *ArXiv*, abs/2006.12179, 2020.

[GBWD+18] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, jan 2018.

[GDH+91] Valerie J. Gillet, Geoffrey M. Downs, John D. Holliday, Michael F. Lynch, and Winfried Dethlefsen. Computer storage and retrieval of generic chemical structures in patents. 13. reduced graph generation. *Journal of Chemical Information and Computer Sciences*, 31(2):260–270, May 1991.

[GGG20] Johannes Gasteiger, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. In *International Conference on Learning Representations*, 2020.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Mathematical Sciences Series. Freeman, 1979.

[GJ22] Hongyang Gao and Shuiwang Ji. Graph u-nets. *IEEE Transactions on Pattern Analysis amp; Machine Intelligence*, 44(09):4948–4960, sep 2022.

[GSR+17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[GWB03] Valerie J. Gillet, Peter Willett, and John Bradshaw. Similarity searching using reduced graphs. *Journal of Chemical Information and Computer Sciences*, 43(2):338–345, Mar 2003.

[Hay09] Simon S. Haykin. *Neural Networks and Learning Machines*. Pearson International Edition. Pearson, 2009.

[HDBDJ14] Martin T. Hagan, Howard B. Demuth, Mark H. Beale, and Orlando De Jesús. *Neural Network Design*. Martin Hagan, 2014.

[HFZ+20] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[HLG+20] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *International Conference on Learning Representations (ICLR)*, 2020.

[Hor91] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[HSK+12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[HTP+18] Truong Son Hy, Shubhendu Trivedi, Horace Pan, Brandon M Anderson, and Risi Kondor. Predicting molecular properties with covariant compositional networks. *J Chem Phys*, 148(24):241745, June 2018.

[HYL17] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[ISM+12] John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. Zinc: A free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768, Jul 2012.

[JBJ18] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pages 2323–2332. PMLR, 2018.

[JEP+21] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon Kohl, Andrew Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:1–11, 08 2021.

[JSL+22] Zewei Ji, Runhan Shi, Jiarui Lu, Fang Li, and Yang Yang. Relmole: Molecular representation learning based on two-level graph similarities. *Journal of Chemical Information and Modeling*, 62(22):5361–5372, Nov 2022.

[JYBJ19] Wengong Jin, Kevin Yang, Regina Barzilay, and Tommi S. Jaakkola. Learning multimodal graph-to-graph translation for molecule optimization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KO23] Apakorn Kengkanna and Masahito Ohue. Enhancing model learning and interpretation using multiple molecular graph representations for compound property and activity prediction. *arXiv preprint arXiv:2304.06253*, 2023.

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[KZL+22] Yue Kong, Xiaoman Zhao, Ruizi Liu, Zhenwu Yang, Hongyan Yin, Bowen Zhao, Jinling Wang, Bingjie Qin, and Aixia Yan. Integrating concept of pharmacophore with graph neural networks for chemical property prediction and interpretation. *Journal of Cheminformatics*, 14(1):52, Aug 2022.

[Lan16] Greg Landrum. Rdkit: Open-source cheminformatics [computer software]. Online: `http://www.rdkit.org`, 2016.

[LGD+19] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International Conference on Machine Learning*, 2019.

[LL22] Pan Li and Jure Leskovec. The expressive power of graph neural networks. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 63–98. Springer Singapore, Singapore, 2022.

[Lou20] Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020.

[MBHSL19a] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[MBHSL19b] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019.

[MBM+17] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, Los Alamitos, CA, USA, jul 2017. IEEE Computer Society.

[McC89] Donna McClish. Analyzing a portion of the roc curve. *Medical decision making : an international journal of the Society for Medical Decision Making*, 9:190–5, 08 1989.

[MRF+19] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4602–4609, Jul. 2019.

[MSRR19] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673. PMLR, 2019.

[Nie15] Michael A. Nielsen. Neural networks and deep learning. *Determination Press*, 2015.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[RD98] Matthias Rarey and J Scott Dixon. Feature trees: A new molecular similarity measure based on tree matching. *J. Comput. Aided Mol. Des.*, 12(5):471–490, 1998.

[SGT+09] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[SP14] Bhavanari Satyanarayana and Kuncham Syam Prasad. Discrete mathematics and graph theory. *PHI Learning Pvt. Ltd.*, 2014.

[Spi64] Leonard Spialter. The atom connectivity matrix (acm) and its characteristic polynomial (acmcp). *Journal of Chemical Documentation*, 4(4):261–269, Oct 1964.

[SWBZ06] Nikolaus Stiefl, Ian Watson, Knut Baumann, and Andrea Zaliani. Erg: 2d pharmacophore descriptions for scaffold hopping. *Journal of chemical information and modeling*, 46:208–20, 04 2006.

[SYS+20] Jonathan M. Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M. Donghia, Craig R. MacNair, Shawn French, Lindsey A. Carfrae, Zohar Bloom-Ackermann, Victoria M. Tran, Anush Chiappino-Pepe, Ahmed H. Badran, Ian W. Andrews, Emma J. Chory, George M. Church, Eric D. Brown, Tommi S. Jaakkola, Regina Barzilay, and James J. Collins. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702.e13, 2020.

[SYZ15] Wanhua Su, Yan Yuan, and Mu Zhu. A relationship between the average precision and the area under the roc curve. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, ICTIR

'15, page 349–352, New York, NY, USA, 2015. Association for Computing Machinery.

[Tri83] Nenad Trinajstić. *Chemical Graph Theory*. Number v. 2 in Chemical Graph Theory. CRC Press, 1983.

[TSS92] Yoshimasa Takahashi, Masayuki Sukekawa, and Shinichi Sasaki. Automatic identification of molecular similarity using reduced-graph representation of chemical structure. *Journal of Chemical Information and Computer Sciences*, 32(6):639–643, Nov 1992.

[VCC+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[vS10] Maarten van Steen. *Graph Theory and Complex Networks: An Introduction*. Maarten van Steen, 2010.

[WL68] Boris Weisfeiler and A. Leman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *"Nauchno-Technicheskaya Informatsia"*, 9, 1968.

[WLLZ18] Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. Cross-lingual knowledge graph alignment via graph convolutional networks. pages 349–357, October-November 2018.

[WLX+20] Yongji Wu, Defu Lian, Yiheng Xu, Le Wu, and Enhong Chen. Graph convolutional networks with markov random field reasoning for social spammer detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):1054–1061, Apr. 2020.

[WRF+18] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chem. Sci.*, 9:513–530, 2018.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *International Conference on Learning Representations*, 2019.

Acronyms

AUC-PRC The Area Under the Precision Recall Curve. 21, 37, 39, 41

AUC-ROC The Area Under the Receiver Operating Characteristic. 21, 37, 39, 41

BCEL BCE with logits loss. 18

ErG Extended Reduced Graph. iii, v, vii, xi, 2, 3, 9–11, 29, 30, 33, 34, 39–41, 45

ErG-Only Our model using only the Extended reduced graph representation. 33, 34, 39–43

FT Feature Tree. iii, v, xi, 1–3, 10, 12, 13, 25, 26, 29, 30, 33, 34, 39–42, 45, 46

FT-Only Our model using only the Feature tree representation. 33, 34, 39–43

GAT Graph Attention Network. 33, 42

GatedGCN Residual Gated Graph Convolution Network. 33, 42

GatedGCN-E Residual Gated Graph Convolution Network using edge atrributes. 33, 41–43

GCN Graph Convolutional Network. 33, 42

GCN-E Graph Convolutional Network using edge atrributes. 33, 41, 43

GIN Graph Isomorphism Network. 33, 35, 42

GIN-E Graph Isomorphism Network using edge atrributes. 33, 35, 40, 42, 43

GNNs Graph Neural Networks. iii, 1, 2, 10, 16, 21, 22, 24–27, 29, 34, 40–43

GraphSAGE Graph Sample and Aggregate. 33, 42

HIMP Hierarchical Inter-Message Passing scheme. iii, vii, ix, xi, 1, 2, 25–27, 29, 30, 33–35, 39–43, 45

MAE Mean absolute error. 18, 21, 36, 40

MLP Multilayer perceptron. 16, 20

MoNet Monti Network. 33, 42

NGF Neural Graph Fingerprint. 33, 39, 40

NN Neural Network. 13, 14, 16–18, 20, 21

RAW+ERG Our model using the raw molecular graph with Extended reduced graph representation with inter-message passing. 34, 40–43, 45

RAW+FT Our model using the raw molecular graph with Feature tree representation with inter-message passing. 34, 40–43, 45

RAW+FT+ERG Our model using the raw molecular graph with Feature tree and Extended reduced graph representations with inter-message passing. 34, 39–43, 45

RAW-Only Our model using only the raw molecular graph. 33, 34, 40–43, 45

RAW||ERG Our model using the raw molecular graph with Extended reduced graph representation without inter-message passing. 34, 40–43, 45

RAW||FT Our model using the raw molecular graph with Feature tree representation without inter-message passing. 34, 40–43, 45

RAW||FT||ERG Our model using the raw molecular graph with Feature tree and Extended reduced graph representations without inter-message passing. 34, 40–43

RG-MPNN Reduced-Graph Message-Passing Neural Network. 33, 39, 40

RP-NGF Relation Pooling Neural Graph Fingerprint. 33, 39, 40