A Theoretical and Empirical Investigation into the Equivalence of Graph Neural Networks and the Weisfeiler-Leman Algorithm

From the faculty of Mathematics, Physics, and Computer Science approved for the purpose of obtaining the academic degree of Bachelor of Sciences.

Eric Tillmann Bill

Supervision:
Prof. Dr. rer. nat. Christopher Morris

Informatik 6
RWTH Aachen University

Contents

1. Introduction

This work aims to shed light on the representations learned by Graph Neural Networks (GNNs). In this section, we will discuss the prevalence of graphs and the crucial role GNNs play in analyzing them. We will delve into the methods we use to gain insights and highlight the significance of our approach. Lastly, we will provide an overview of the structure of this work.

1.1. Motivation

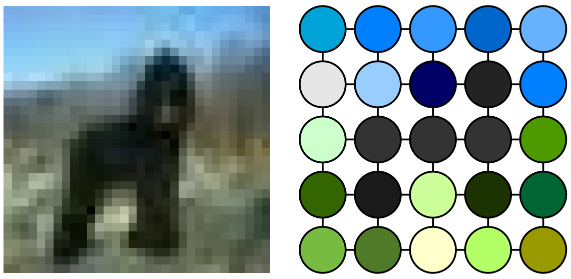
Graphs are ubiquitous in various fields of life. Despite not always being explicitly identified as such, the graph data model's flexibility and simplicity make it an effective tool for modeling a diverse range of data. Examples of graph modeling applications include unexpected instances, such as modeling text or images as a graph, as well as more complex instances like chemical molecules, citation networks, or connectivity encodings of the World Wide Web Morris et al. [2020], Scarselli et al. [2009].

Although machine learning has achieved remarkable success with image classification (e.g., Zoph et al. [2018], He et al. [2016]) and text generation (e.g., Radford et al. [2019], Brown et al. [2020]) in the last decade, the lack of a significant breakthrough in machine learning for graphs can be attributed to the graph data model's inherent flexibility and simplicity. While, for example, an image classifier constrains its input data to be a grid-like image or a text generator expects its input to be a linear sequence of words, machine learning models working on graphs cannot leverage any constraints on the format or size of their input graphs without limiting their generality.

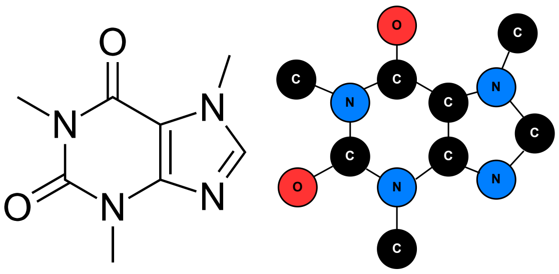
To put this flexibility of the graph data model into perspective and give an idea of how ubiquitous graphs are in various fields, we refer back to the examples of image classifiers and text generators and demonstrate how seemingly natural the graph data model can encode their input data. For example, images can be encoded by a graph, such that each pixel of the image corresponds to a node in the graph holding its color value, and each node is connected to its neighboring pixel nodes. Similarly, for sequential data like text files, one can encode a directed graph where each word in this file is represented as a node with the word as a feature and connected outgoingly to the next following word. See Figure 1 for an illustrative example of these encodings.

In recent years, a significant amount of research has been conducted to investigate Graph Neural Networks (GNNs). Among the most promising approaches are those utilizing the message-passing architecture, which was introduced by Scarselli et al. [2009] and Gilmer et al. [2017]. Empirically, this framework has demonstrated strong performance across various applications Kipf and Welling [2017], Hamilton et al. [2017], Xu et al. [2019]. However, its expressiveness is limited, as has been proved by the works of Morris et al. [2019], as well as Xu et al. [2019]. These works establish a connection to the commonly used Weisfeiler-Leman[1] algorithm (1-WL), originally proposed by Weisfeiler and Leman [1968] as a simple heuristic for the graph isomorphism problem. In particular, it has been proven that a GNN based on the message-passing architecture can, at most, be as good as the 1-WL algorithm in distinguishing non-isomorphic graphs. Furthermore, the 1-WL method demonstrates numerous similarities with the fundamental workings of the GNN architecture. It is, therefore, commonly believed

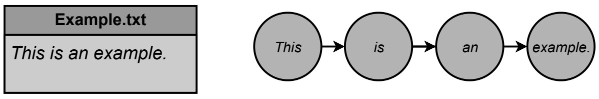
[1]Based on https://www.iti.zcu.cz/wl2018/pdf/leman.pdf, we will use the spelling "Leman" here, as requested by A. Leman, the co-inventor of the algorithm.



(a) Example for an image of a dog.[2]



(b) Example for the molecule caffeine.[3]



(c) Example for a text file.

Figure 1.: Here are some examples of how graphs can be used to encode information in a variety of domains. Please note that these examples are just a sample, and in actual practice, more detailed encodings are usually utilized to capture additional information.[4]

that both methods are, to some extent, equivalent in their capacity to capture information in a graph.

Footnotes are wrong!

more colorful and downscaled example :(

1.2. Research Questions & Contributions

Test

1.3. Methology

In this work, we introduce a novel framework, which we coined "1-WL+NN," which involves applying the 1-WL algorithm to an input graph and further processing the resulting information using a feedforward neural network. Thereby, we obtain a trainable framework suited for all kinds of graph-related tasks, such as graph classification, node regression, and more. We will prove later on that both frameworks, 1-WL+NN and GNN, are theoretically equivalent, such that each function computed by a 1-WL+NN can also be computed by a GNN model and vice versa. With this framework in hand, we can investigate the representations learned by a GNN.

The interesting property of this framework compared to GNNs, which is also the original idea that inspired this work, is the fundamental difference in how both frameworks learn and optimize themselves when applied to a specific task. Take, for example, an arbitrary classification task. While the first part of a 1-WL+NN model starts by applying the 1-WL algorithm to its input graph and retrieves a highly informative representation of this graph, the second part, the learnable feedforward neural network, must find common patterns in this very detailed representation that coincides with the classes of the task, such that the model

[2]The image is from the CIFAR-10 collection made available by Krizhevsky et al. [2009].

[3]The illustration of the skeletal formula of caffeine is taken from https://commons.wikimedia.org.

[4]All graphics were created using the free open source platform https://www.draw.io.

makes good predictions. In comparison, while a GNN is theoretically as expressive as the 1-WL algorithm, we imagine that such a model first leans to find common patterns as early as possible in the input graph and will drop irrelevant information as soon as possible before making a class prediction.

To put both learning behaviors into perspective: while the 1-WL+NN is given a maximally informative representation and needs to find the important information to make a good prediction, a GNN works the other way around and needs to learn how it constructs its own informative representation of the input graph and how to leverage this information for making a prediction. Therefore, we expect 1-WL+NN models to perform sufficiently well on their training data but expect poor generalization capabilities due to the too-informative representations computed by the 1-WL algorithm. In contrast, we expect more promising generalization capabilities of GNNs in comparison since they optimize for the best representation of their input graphs and might leverage this information more efficiently.

We will use this novel framework and various empirical experiments to compare both frameworks on multiple datasets to establish a deeper understanding of the representations learned by GNNs.

1.4. Outline

For the ease of readability, we split this work into two parts. The first part investigates and establishes the theoretical equivalence between the frameworks of 1-WL+NN and GNNs. In contrast, the second part discusses our different experiments and their empirical results and insights into GNNs.

In detail, this work starts with Section 2, where we discuss related work, milestones in GNNs over the past decade, essential properties of the 1-WL algorithm, and a subset of interesting connections between GNNs and the 1-WL algorithm. Afterward, we will start with Part I, which begins with Section 3. Here, we will introduce formal definitions for both frameworks, as well as a set of notations we will use throughout the theoretical part. Afterward, in Section 4, we will introduce four theorems that each present a connection between both frameworks and combined prove the equivalence of both frameworks. Finally, we will prove each theorem individually after another in corresponding subsections.

The second part will deal with ...

2. Background and Related Work

In this section, we will briefly introduce the foundation of our research by explaining the origins of each method, mentioning important recent advances, and providing a brief overview of the connections between them.

2.1. Weisfeiler-Leman Algorithm

The (1-dimensional) Weisfeiler-Leman algorithm (1-WL), proposed by Weisfeiler and Leman [1968], was initially designed as a simple heuristic for the *graph isomorphism problem*, but due to its interesting properties, its simplicity, and its good performance, the 1-WL algorithm gained a lot of attention from researchers across many fields. One of the most noticeable properties is that the algorithm color codes the nodes of the input graph in such a way that in each iteration, each color encodes a learned local substructure.

It works by coloring all nodes in each iteration the same color that fulfills two properties: 1. the nodes already share the same color, and 2. the count of each color of their direct neighbors is equal. The algorithm continues as long as the number of colors changes in each iteration. For determining whether two graphs are non-isomorphic, the heuristic is applied to both graphs simultaneously. It concludes that the graphs are non-isomorphic as soon as the number of occurrences of a color differs between them. We present a more formal definition of the algorithm in the following part in subsection 3.3.

Since the *graph isomorphism problem* is difficult to solve due to the best known complete algorithm only running in deterministic quasipolynomial time (Babai [2016]), the 1-WL algorithm, running in deterministic polynomial time, cannot solve the problem completely. Moreover, Cai et al. [1992] constructed counterexamples of non-isomorphic graphs that the heuristic fails to distinguish, e.g., see Figure 2. However, following the work of Babai and Kucera [1979], this simple heuristic is still quite powerful and has a very low probability of failing to distinguish non-isomorphic graphs when both graphs are uniformly chosen at random as the number of nodes tends to infinity.

To overcome the limited expressiveness of the 1-WL algorithm, it has been generalized to the k-dimensional Weisfeiler-Leman algorithm (k-WL) by Babai [1979, 2016], as well as Immerman and Lander [1990][5]. This version works with k-tuples over the k-ary Cartesian product of the set of nodes. Interestingly, this created a hierarchy for the expressiveness of determining non-isomorphism, such that for all $k \in \mathbb{N}$ there exists a pair of non-isomorphic graphs that can be distinguished by the $(k+1)$-WL but not by the k-WL (Cai et al. [1992]).

2.2. Graph Neural Networks

The utilization of machine learning techniques, previously proven effective in various domains, for graph analysis has been a well-established topic in the literature for several decades. However, researchers faced challenges in effectively adapting these techniques to graphs of diverse sizes and complexities in the early stages. Notably, the seminal works by Sperduti (1997), Scarselli (2008), and Micheli (2009) emerged as prominent examples of successful applications in this regard.

The idea of leveraging machine learning techniques, previously proven effective in various domains, for graph-related tasks has been a well-established topic in the literature for the past decades. However, in the early stages, researchers faced challenges in effectively adapting these techniques to work on graphs of arbitrary sizes and complexities. Notably, the works by Sperduti and Starita [1997], Scarselli et al. [2008], and Micheli [2009] were the first prominent examples of successful applications in this regard.

However, it was not until the emergence of more advanced models that the scientific community truly recognized the significance and potential of Graph Neural Networks (GNNs). Noteworthy among these advancements were the work of Duvenaud et al. [2015], who introduced a differentiable approach for generating unique fingerprints of arbitrary graphs, as well as Li et al. [2015], who applied gated recurrent units to capture graphs of various sizes, while Atwood and Towsley [2016] utilized diffusional convolutions for the same purpose. Of particular significance, however, were the contributions of Bruna et al. [2013], Defferrard et al. [2016] and Kipf and

[5]In Babai [2016] on page 27, László Babai explains that he, together with Rudolf Mathon, first introduced this algorithm in 1979. He adds that the work of Immerman and Lander [1990] introduced this algorithm independently of him.

Welling [2017], which extended the concept of convolution from its traditional application on images to the domain of arbitrary graphs.

After the early success of these GNNs, Gilmer et al. [2017] introduced a unified architecture for GNNs. The authors observed a recurring pattern in how information is exchanged and processed among many of these works, including many mentioned in the paragraph above. Leveraging these observations, Gilmer et al. [2017] devised the message-passing architecture as a generalized framework for GNNs. Models using this architecture can be referred to as Message-Passing-Neural-Network (MPNN); however, throughout this thesis, we will use the term GNN and MPNN interchangeably, as the focus of this thesis is solely on the message-passing architecture. This architecture uses the input graph as its basis for computation and computes new node features for the graph in each layer. Each new node feature is derived by aggregating the nodes and neighboring node features. After applying each layer of a GNN model, a representation of the entire graph is obtained by applying a pooling function (see YingMorris2018). This representation is then further processed by common machine learning techniques like a multilayer perceptron for the final output. We will present a more formal definition of this architecture in the following part; however, important to note is that the information exchange in the graph across nodes is limited to a one-hop neighbor per layer.

With this general framework and the empirical success of some models using this message-passing architecture, the question of how expressive models based on this architecture can be gained a lot of attention in the scientific community. Many papers immediately established connections to the 1-WL algorithm, among the most prominent being Morris et al. [2019] and Xu et al. [2019]. These connections seem natural, as both methods share similar properties in terms of how they process graph data. Most strikingly, both methods never change the graph structurally since they only compute new node features in each iteration. Moreover, both methods use a one-hop neighborhood aggregation as the basis for computing the new node feature. Following this intuition, Morris et al. [2019] as well as Xu et al. [2019] showed that the expressiveness of GNN is upper-bounded by the 1-WL in terms of distinguishing non-isomorphic graphs. Moreover, Morris et al. [2019] proposed a new k-GNN architecture that operates over the set of subgraphs of size k. Interestingly, Geerts [2020] as well as Grohe [2017] have shown that the proposed hierarchy over $k \in \mathbb{N}$ is equivalent to the k-WL hierarchy in terms of its ability to distinguish non-isomorphic graphs, i.e., if there is a k-GNN that can distinguish two non-isomorphic graphs, it is equivalent to say that the k-WL algorithm can also distinguish these graphs.

Although there are other modifications of the message-passing architecture besides the theoretical concept of k-GNN to increase the expressiveness of GNNs in terms of distinguishing non-isomorphism, e.g., using node identifiers Vignac et al. [2020], adding random node features Sato et al. [2021], Abboud et al. [2020], adding directed flows Beaini et al. [2021] and many more. Relatively few works have been published that attempt to understand the representation learned from a standard GNN.

Notable works include Nikolentzos et al. [2023], where the author, in addition to the normal learning process, optimized GNNs to preserve a notion of distance in their representation and examined the effectiveness of GNNs in utilizing such representations. However, their insights can only be applied to these specially trained GNNs, not standard ones. In another publication, Nikolentzos et al. [2023] presented mathematical proof and empirical confirmation showing how much structural information is encoded by modern GNN models. The research highlights that GNN models like DGCNN (Zhang et al. [2018]) and GAT (Veličković et al. [2017]) encode all nodes with the same feature vector, while in contrast, models like GCN (Kipf and Welling

[2017]) and GIN (Xu et al. [2019]) encode nodes after k layer of message-passing with features that relate with the number of walks of length k form each node, disregarding the local structure within the nodes are contained.

Part I.

Theoretical Equivalence

3. Preliminaries

First, we introduce a couple of notions and definitions that will be used throughout this thesis. In particular, the definitions will be crucial in the following section containing the proofs. We start with general notations, introduce a general graph definition, and familiarize the reader with the Weisfeiler-Leman algorithm. We will introduce each framework independently, first the 1-WL+NN and then GNN. In the end, we will briefly introduce important properties of collections of functions computed by both methods.

3.1. General Notation

Let $\mathbb{N}$ denote the set of natural numbers such that $\mathbb{N} := \{0, 1, 2, \ldots\}$. By $[n]$, we denote the set $\{1, \ldots, n\} \subset \mathbb{N}$ for each $n \in \mathbb{N}$. Further, with $\{\!\{\ldots\}\!\}$ we denote a multiset formally defined as a 2-tuple (X, m), where X is a set of all unique elements and $m : X \to \mathbb{N}_{\geq 1}$ a mapping that maps each element in X to the number of its occurrences in the multiset.

3.2. Graphs

A graph G is a 3-tuple $G := (V, E, l)$ that consists of the set of all nodes V, the set of all edges $E \subseteq V \times V$ and a label function $l : M \to \Sigma$ with M being either $V, V \cup E$ or E and $\Sigma \subset \mathbb{N}$ a finite alphabet. Moreover, let $\mathcal{G}$ be the set of all finite graphs. Note that our definition of the labeling function allows graphs with labels either on the nodes only, on the edges only, or on both the nodes and the edges. In some cases we refer to the values assigned by l as features instead of labels, however this is usually only the case when Σ is multidimensional. In addition, although we have defined it this way, the labeling function is optional, and in cases where no labeling function is given, we add the trivial labeling function $f_1 : V(G) \to \{1\}$.

Further, G can be either directed or undirected, depending on the definition of E, where $E \subseteq \{(v, u) \mid v, u \in V\}$ defines a directed and $E \subseteq \{(v, u), (u, v) \mid v, u \in V, v \neq u\}$ such that for every $(v, u) \in E$ also $(u, v) \in E$ defines an undirected graph. Additionally, we will use the notation $V(G)$ and $E(G)$ to denote the set of nodes of G and the set of edges of G respectively, as well as l_G to denote the label function of G. With $\mathcal{N}(v)$ for $v \in V(G)$ we denote the set of neighbors of v defined as $\mathcal{N}(v) := \{u \mid (u, v) \in E(G)\}$.

A coloring of a Graph G is a function $C : V(G) \to \mathbb{N}$ that assigns each node in the graph a color (here a positive integer). Further, a coloring C induces a partition P on the set of nodes, for which we define C^{-1} being the function that maps each color $c \in \mathbb{N}$ to its class of nodes with $C^{-1}(c) = \{v \in V(G) \mid C(v) = c\}$. In addition, we define $h_{G,C}$ as the histogram of graph G with coloring C, that maps every color in the image of C under $V(G)$ to the number of occurrences. In detail, $\forall c \in \mathbb{N} : h_{G,C}(c) := |\{v \in V(G) \mid C(v) = c\}| = |C^{-1}(c)|$

Permutation-invariance and -equivariance

We use S_n to denote the symmetric group over the elements $[n]$ for any $n \in \mathbb{N}_{\geq 1}$. S_n consists of all permutations over these elements. Let G be a graph with $V(G) = [n]$, applying a permutation $\pi \in S_n$ on G, is defined as $G_\pi := \pi \cdot G$ where $V(G_\pi) = \{\pi(1), \ldots, \pi(n)\}$ and $E(G_\pi) = \{(\pi(v), \pi(u)) \mid (v, u) \in E(G)\}$. We will now introduce two key concepts for classifying functions on graphs.

Definition 1 (Permutation Invariant)**.** Let $f : \mathcal{G} \to \mathcal{X}$ be an arbitrary function, then f is *permutation-invariant* if and only if for all $G \in \mathcal{G}$ where $n_G :=| V(G) |$ and for every $\pi \in S_{n_G}$: $f(G) = f(\pi \cdot G)$.

Definition 2 (Permuation Equivariant)**.** Let $f : \mathcal{G} \to \mathcal{X}$ be an arbitrary function, then f is *permuation-equivariant* if and only if for all $G \in \mathcal{G}$ where $n_G :=| V(G) |$ and for every $\pi \in S_{n_G}$: $f(G) = \pi^{-1} \cdot f(\pi \cdot G)$.

3.3. Weisfeiler and Leman Algorithm

The Weisfeiler-Leman algorithm consists of two main parts, first the coloring algorithm and second the graph isomorphism test. We will introduce them in this section.

The Weisfeiler-Leman graph coloring algorithm

The 1-WL algorithm computes a node coloring of its input graph in each iteration. A color for a node is computed using only the coloring of its neighbors and the node itself. The algorithm will continue as long as it has not converged, and returns the final coloring of the graph.

Definition 3 (1-WL Algorithm)**.** Let $G = (V, E, l)$ be a graph, then in each iteration i, the 1-WL computes a node coloring $C_i : V(G) \to \mathbb{N}$. In iteration $i = 0$, the initial coloring is $C_0 = l$ or if l is non-existing $\forall v \in V(G) : C_0(v) = c$ for an arbitrary constant $c \in \mathbb{N}$. For $i > 0$, the algorithm assigns a color to $v \in V(G)$ as follows:

$$C_i(v) = \mathsf{RELABEL}(C_{i-1}(v),\ \{\!\!\{C_{i-1}(u) \mid u \in \mathcal{N}(v)\}\!\!\}),$$

where RELABEL injectively maps the above pair to a unique, previously not used, natural number. The algorithm terminates when the number of colors between two iterations does not change, meaning the algorithm terminates after iteration i if the following condition is satisfied:

$$\forall v, w \in V(G) : C_i(v) = C_i(w) \iff C_{i+1}(v) = C_{i+1}(w).$$

Upon terminating we define $C_\infty := C_i$ as the stable coloring, such that 1-WL$(G) := C_\infty$.

The colorings computed in each iteration always converge to the final one, such that the algorithm always terminates. In more detail, Grohe [2017] showed that it always holds after at most $|V(G)|$ iterations. For an illustration of this coloring algorithm, see Figure 3. Moreover, based on the work of Paige and Tarjan [1987] about efficient refinement strategies, Cardon and Crochemore [1982] proved that the stable coloring C_∞ can be computed in time $\mathcal{O}(|V(G)| + |E(G)| \cdot \log |V(G)|)$.

The Weisfeiler-Leman Graph Isomorphism Test

Definition 4 (1-WL Isomorphism Test)**.** To determine if two graphs $G, H \in \mathcal{G}$ are non-isomorphic ($G \not\simeq H$), one applies the 1-WL coloring algorithm on both graphs "in parallel" and checks after each iteration if the occurrences of each color are equal, else the algorithm would terminate and conclude non-isomorphic. Formally, the algorithm concludes non-isomorphic in iteration i if there exists a color c such that:

$$|\{v \in V(G) \mid c = C_i(v)\}| \neq |\{v \in V(H) \mid c = C_i(v)\}|.$$

Note that this test is only sound and not complete for the *graph isomorphism problem*. Counterexamples where the algorithm fails to distinguish non-isomorphic graphs can be easily constructed, see Figure 2 which was discovered and proven by Cai et al. [1992].

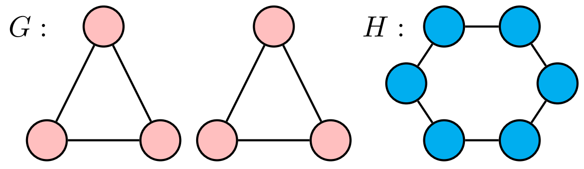


Figure 2.: An example of two graphs G and H that are non-isomorphic but cannot be distinguished by the 1-WL isomorphism test.

3.4. 1-WL+NN

As seen in the previous section, the 1-WL algorithm is quite powerful in identifying substructures. With the 1-WL+NN framework, we define functions that utilize this structural information to derive further application-specific insights. We do this by combining well-known machine learning techniques and the algorithm.

Definition 5 (1-WL+NN)**.** We say a function $\mathcal{B}$ working over a set of graphs $\mathcal{X} \subseteq \mathcal{G}$ is computable by 1-WL+NN, if it can be decomposed into a multilayer perceptron MLP, an encoding function f_{enc} and the 1-WL algorithm. In particular, when dealing with graph task $\mathcal{B}$ can be decomposed as:

$$\mathcal{B} : \mathcal{X} \to \mathbb{R},\ G \mapsto \text{MLP} \circ f_{\text{enc}}(\{\!\!\{\text{1-WL}(G)(v) \mid v \in V(G)\}\!\!\}),$$

where 1-WL(G) is the coloring computed by the 1-WL algorithm when applied on G. For node or edge tasks we define it as below:

$$\mathcal{B} : \mathcal{X} \to \mathcal{Y},\ G \mapsto G' \coloneqq (V(G),\ E(G),\ \text{MLP} \circ \text{1-WL}(G)),$$

where a graph G is mapped to a structurally identical graph G' with the key difference being that the label function is replaced with the coloring computed by the 1-WL algorithm and further processed by a MLP.

As a concrete example of a collection of functions computable by 1-WL+NN we will introduce the collection of functions working from $\mathcal{X}$ to $\mathbb{R}$ that use the so called *counting-encoding* function as its encoding function. We will define this particular encoding function in the following:

Definition 6 (Counting Encoding Functions)**.** The *counting-encoding* function $f_{\text{count},k}$ is parametrized by a $k \in \mathbb{N}$ and works as follows:

$$f_{\text{count},k}(\{\!\!\{C(v) \mid v \in V(G)\}\!\!\}) \coloneqq \begin{bmatrix} |\{v \in V(G) \mid C(v) = 1\}| \\ \vdots \\ |\{v \in V(G) \mid C(v) = k\}| \end{bmatrix} \in \mathbb{N}^k.$$

It maps the occurrences of the colors $c \in [k]$ of its input to a vector over $\mathbb{N}^k$, such that each i.th component of this vector encodes the number of occurrences of the color i in the input.

To illustrate how this encoding function works and why we coined it *counting-encoding*, we will quickly introduce an example graph G. In Figure 3, we give a visual representation of G and its stable coloring after applying the 1-WL algorithm to it. The *counting-encoding* function $f_{\text{count},k}$ counts through all colors $i \in [k]$ and sets each i.th component of the output vector to the number of occurrences in the final coloring. Therefore, the respective color histogram $h_{G,C_\infty} = \{\{2, 2, 3, 4\}\}$ of G is being mapped to $(0, 2, 1, 1)^T \in \mathbb{N}^4$ by $f_{\text{count},4}$ and to $(0, 2)^T \in \mathbb{N}^2$ by $f_{\text{count},2}$ for example, since color 2 appears two times, while color 3 and 4 occur only once.

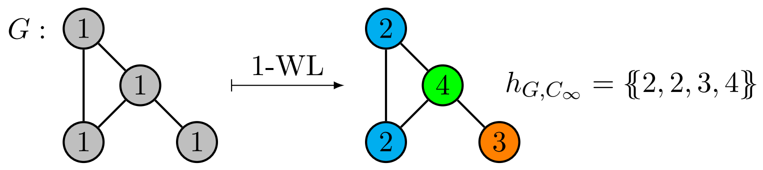


Figure 3.: An example of the final coloring computed by applying the 1-WL algorithm on the graph G. The graph G consists of 4 nodes with all their labels being initially set to 1. Note that each label corresponds to a color, which we have also plotted for illustration purposes.

3.5. Graph Neural Networks (Message Passing)

A Graph Neural Network (GNN) is a composition of multiple layers, where each layer computes a new feature for each node and edge. The GNN layer thus technically obtains a new graph that is structurally identical to the previous one, but contains new feature information. After an input graph has been passed through all layers, there can be an additional final function, aggregating the computed information into a fixed size output. With this, it is possible to apply a GNN to every graph, regardless of its size, as the "computation" will only take place on the nodes and edges of the graph.

Note that in the following we will restrict the definition to only consider node features, however, one can easily extend it to also include edge features.

Definition 7 (Graph Neural Network)**.** Let $G = (V, E, l)$ be an arbitrary graph. A Graph Neural Network (GNN) is a composition of multiple layers where each layer t is represented by a function $f^{(t)}$ that works over the set of nodes $V(G)$. To begin with, we need a function $f^{(0)} : V(G) \to \mathbb{R}^{1\times d}$ that is consistent with l, that translates all labels into a vector representation. Further, for every $t > 0$, $f^{(t)}$ is of the format:

$$f^{(t)}(v) = f_{\text{merge}}^{W_{1,t}}(f^{(t-1)}(v),\ f_{\text{agg}}^{W_{2,t}}(\{\{f^{(t-1)}(w) \mid w \in \mathcal{N}(v)\}\})),$$

where $f_{\text{merge}}^{W_{1,t}}$ and $f_{\text{agg}}^{W_{2,t}}$ are arbitrary differentiable functions with $W_{1,t}$ and $W_{2,t}$ their respective parameters. Additionally, $f_{\text{agg}}^{W_{2,t}}$ has to be permuation-invariant.

Depending on the objective, whether the GNN is tasked with a graph or a node task, the last layer differs. In the case of graph tasks, we add a permutation-invariant aggregation function to the end, here called READOUT, working as follows:

$$\mathsf{READOUT}(\{\{f^{(t)}(v) \mid v \in V(G)\}\}),$$

that aggregates over every node and computes a fixed-size output vector for the entire graph, e.g. a label for graph classification. In order to ensure that we can train the GNN in an end-to-end fashion, we require $\mathsf{READOUT}$ to be also differentiable. Let $\mathcal{A}$ be an instance of the described GNN framework. Further, let $K \in \mathbb{N}$ be the number of layers of the GNN, $\mathcal{G}$ the set of all graphs, $\mathcal{Y}$ the task-specific output set (e.g. labels of a classification task), then the overall function computed by $\mathcal{A}$ is:

$$\mathcal{A} : \mathcal{G} \to \mathcal{Y},\ G \mapsto \mathsf{READOUT}(\{\!\!\{ f^{(K)}(v) \mid v \in V(G) \}\!\!\}),$$

if $\mathcal{A}$ is configured for a graph task, otherwise:

$$\mathcal{A} : \mathcal{G} \to \mathcal{Y},\ G \mapsto G' \coloneqq (V(G),\ E(G),\ f^{(K)}),$$

where a graph G is mapped to a structurally identical graph G' with the key difference being the label function is $f^{(K)}$.

Note that, as we require all aggregation functions to be permutation-invariant, the total composition $\mathcal{A}$ is permutation-invariant, and with similar reasoning, it is also differentiable. This enables us to train $\mathcal{A}$ like any other machine learning method in an end-to-end fashion, regardless of the underlying encoding used for graphs. This definition and use of notation are inspired by Morris et al. [2019] and Xu et al. [2019].

To demonstrate what kind of functions are typically used, we provide functions used by Hamilton et al. [2017] for a node classification:

$$\begin{aligned} f_{\text{merge}}^{W_{1,t}}(v) &= \text{ReLU}(W_{\text{merge}} \cdot \mathsf{concat}(f^{(t-1)}(v),\ f_{\text{agg}}^{W_{2,t}}(v))) \\ f_{\text{agg}}^{W_{2,t}}(v) &= \mathsf{MAX}(\{\text{ReLU}(W_{\text{pool}} \cdot f^{(t-1)}(u) + b) \mid u \in \mathcal{N}(v)\}) \end{aligned}$$

where ReLU is a rectified linear unit element wise activation function, MAX the element-wise max operator; W_{merge}, W_{pool} are trainable matrices, b a trainable vector and concat the concatenation function.

Model	**Update Equation**
GCN	$h_v^{(k)} = \text{ReLU}\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{W^k h_u^{(k-1)}}{\sqrt{(1+d(v))(1+d(u))}}\right)$

Table 1.: Overview of attacks on FL architectures

3.6. General Definitions

In this section, we introduce a formal definition of multilayer perceptron as it is required in a later proof, as well as the $\simeq_{1\text{WL}}$ relation. Additionally, two very important properties for collections of functions.

Definition 8 (Multilayer Perceptron)**.** Multilayer perceptrons are a class of functions from $\mathbb{R}^n$ to $\mathbb{R}^m$, with $n, m \in \mathbb{N}$. In this thesis, we define a multilayer perceptron as a finite sequence, such that a multilayer perceptron MLP is defined as $\text{MLP} \coloneqq (\text{MLP})_{i \in [k]}$ where k is the number of layers. For every $i \in [k]$, the i.th layer of the MLP is the i.th item in the finite sequence $(\text{MLP})_i$. Further, all layers are recursively defined as:

Adapt the notation for sequences $(\text{MLP}_i)_{i \in [k]}$

$$\begin{aligned} (\text{MLP})_0(v) &\coloneqq v \\ (\text{MLP})_{i+1}(v) &\coloneqq \sigma_i(W_i \cdot (\text{MLP})_i(v) + b_i), \quad \forall i \in [k-1] \end{aligned}$$

where σ_i is an element wise activation function, W_i is the weight matrix and b_i the bias vector of layer i. Note, that for each W_i, the succeeding W_{i+1} must have the same number of columns as W_i has rows, in order to be well-defined. Similarly, for every layer i, W_i and b_i have to have the same number of rows. Following this definition, when applying a MLP on input $v \in \mathbb{R}^n$ it is $\text{MLP}(v) := (\text{MLP})_k(v)$.

Definition 9 (1-WL Relation)**.** For any graphs $G, H \in \mathcal{X}$ we will denote $G \simeq_{1\text{WL}} H$ if the 1-WL isomorphism test can not distinguish both graphs. Note that due to the soundness of this algorithm, if $G \not\simeq_{1\text{WL}} H$, we always can conclude that $G \not\simeq H$.

The $\simeq_{1\text{WL}}$ relation can further be classified as an equivalence relation, as it is reflexive, symmetric and transitive. With this, we introduce a notation of its equivalence classes. Let $\mathcal{X} \subseteq \mathcal{G}$ and $G \in \mathcal{X}$, then we denote with $\mathcal{X}/{\simeq_{1\text{WL}}}(G) := \{G' \in \mathcal{X} \mid G \simeq_{1\text{WL}} G'\}$ its equivalence class.

Definition 10 (1-WL-Discriminating)**.** Let $\mathcal{C}$ be a collection of permutation invariant functions from $\mathcal{X}$ to $\mathbb{R}$. We say $\mathcal{C}$ is **1-WL-Discriminating** if for all graphs $G_1, G_2 \in \mathcal{X}$ for which the 1-WL isomorphism test concludes non-isomorphic ($G_1 \not\simeq_{1\text{WL}} G_2$), there exists a function $h \in \mathcal{C}$ such that $h(G_1) \neq h(G_2)$.

Definition 11 (GNN-Approximating)**.** Let $\mathcal{C}$ be a collection of permutation invariant functions from $\mathcal{X}$ to $\mathbb{R}$. We say $\mathcal{C}$ is **GNN-Approximating** if for all permutation-invariant functions $\mathcal{A}$ computed by a GNN, and for all $\epsilon \in \mathbb{R}$ with $\epsilon > 0$, there exists $h_{\mathcal{A},\epsilon} \in \mathcal{C}$ such that $\|\mathcal{A} - h_{\mathcal{A},\epsilon}\|_\infty := \sup_{G \in \mathcal{X}} |\mathcal{A}(G) - h_{\mathcal{A},\epsilon}(G)| < \epsilon$

4. Theoretical Connection

This section is the main part of our theoretical investigation of the two frameworks. We will present 4 intriguing theorems, which will be proven separately afterwards. These results will form the basis for the empirical part that follows. The first two theorems will establish an equivalence between the two frameworks when the input set of graphs is finite. The last two theorems will go one step further and establish a connection for continuous functions computed by 1-WL+NN and GNNs and prove a somewhat weaker connection between them.

All the results we will now present focus on graph task functions, since this is the more complex problem, as any node or edge task function can be adapted to a graph task. In the first two theorems, we focus on a finite collection of graphs, which we denote by $\mathcal{X} \subset \mathcal{G}$.

Theorem 12 (Finite Case: "GNN $\subseteq$ 1-WL+NN")**.** Let $\mathcal{C}$ be a collection of functions from $\mathcal{X}$ to $\mathbb{R}$ computable by GNNs, then $\mathcal{C}$ is also computable by 1-WL+NN.

Theorem 13 (Finite Case: "1-WL+NN $\subseteq$ GNN")**.** Let $\mathcal{C}$ be a collection of functions from $\mathcal{X}$ to $\mathbb{R}$ computable by 1-WL+NN, then $\mathcal{C}$ is also computable by GNNs.

With these theorems, we showed the equivalence between both frameworks. Specifically, every function computed by 1-WL+NN is also computable by a GNN, and vice versa. Notice that, we did not leverage any constraints on the encoding of graphs throughout the first two theorems and their corresponding proves, but rather kept it general.

Having established a connection between the two frameworks on a finite subset of graphs, we wanted to further demonstrate the expressive power of 1-WL-Discriminating by investigating a connection between the two frameworks for continuous feature spaces and continuous functions. However, since the graph isomorphism problem is an inherently discrete problem, the 1-WL algorithm is only defined as a discrete and discontinuous function operating on discrete colors, such that extending the definition of the 1-WL algorithm to a continuous function working over continuous values is not very trivial and, to our knowledge, has not yet been widely investigated. We therefore assume in the proofs and the following theorems that there exists a continuous version of the 1-WL algorithm.

We define the set of graphs with continuous features using the following definition:

Definition 14. Let X be a compact subset of $\mathbb{R}$ including 0. We decode graphs with n nodes as a matrix $G \in X^{n \times n}$, where $G_{i,i}$ decodes the label of node i for $i \in [n]$, and $G_{i,j}$ with $i \neq j \in [n]$ decodes an edge from node i to j and a corresponding edge features. Furthermore, we say that there is an edge between node i and j if and only if $G_{i,j} \neq 0$. Additionally, if G encodes an undirected graph, G is a symmetric matrix. For simplicity, we denote $\mathcal{X} := X^{n \times n}$ throughout the next two theorems and their respective proofs.

Theorem 15 (Continuous Case: "GNN $\subseteq$ 1-WL+NN")**.** Let $\mathcal{C}$ be a collection of continuous functions from $\mathcal{X}$ to $\mathbb{R}$ computable by 1-WL+NN. If $\mathcal{C}$ is 1-WL-Discriminating, then there exists a collection of functions $\mathcal{C}'$ computable by 1-WL+NN that is GNN-Approximating.

Theorem 16 (Continuous Case: "1-WL+NN $\subseteq$ GNN")**.** Let $\mathcal{C}$ be a collection of continuous functions from $\mathcal{X}$ to $\mathbb{R}$ that is GNN-Approximating, then $\mathcal{C}$ is also 1-WL-Discriminating.

Since we only wanted to show the expressive power of 1-WL-Discriminating and made the major assumption of the existence of a continuous 1-WL algorithm, we have included the proofs of Theorems 15 and 16 in the Appendix in subsection 2.2.

Immediately from the last theorem follows the corollary:

Corollary 17. There exists a collection $\mathcal{C}$ of function from $\mathcal{X}$ to $\mathbb{R}$ computable by GNNs that is 1-WL-Discriminating.

Proof. The collection of all functions from $\mathcal{X}$ to $\mathbb{R}$ computable by GNNs is trivially GNN-Approximating, such that we can apply Theorem 16 with which the proof concludes. □

By putting both theorems into perspective, we can now argue that even for continuous functions 1-WL+NN and GNNs can compute almost the same functions. Each framework can approximate the other framework arbitrarily well. From this we can conclude that the ability 1-WL-Discriminating is very powerful, so we can assume that 1-WL+NN is sufficiently expressive for the upcoming empirical part.

4.1. Proof of Theorem 12

We will prove Theorem 12 by introducing a couple of small lemmas, which combined prove the theorem. In detail, in Lemma 18 we show the existence of a collection computed by 1-WL+NN that is 1-WL-Discriminating. In Lemmas 19 to 21 we derive properties of 1-WL+NN functions we will use throughout Lemmas 22, 23 and 28 with which we prove the theorem. We took great inspiration for Lemmas 22, 23 and 28 from the proof presented in section 3.1 in the work of Chen et al. [2019].

Lemma 18. There exists a collection $\mathcal{C}$ of functions from $\mathcal{X}$ to $\mathbb{R}$ computable by 1-WL+NN that is 1-WL-Discriminating.

Proof. We define f_c for $c \in \mathbb{N}$ as the encoding functions that returns the number of nodes colored as c. With this we can construct the collection of functions C as follows:

$$C := \{\mathcal{B}_c : \mathcal{X} \to \mathbb{R},\ G \mapsto \mathrm{MLP}_{\mathrm{id}} \circ f_c(\{\!\{\text{1-WL}(G)(v) \mid v \in V(G)\}\!\}) \mid c \in \mathbb{N}\},$$

where $\mathrm{MLP}_{\mathrm{id}}$ is a dummy multilayer perceptron that just returns its input. Since every function $\mathcal{B}_c \in C$ is composed of the 1-WL algorithm, an encoding function and a multilayer perceptron, each function is computable by 1-WL+NN, and consequently also the whole collection.

Let $G_1, G_2 \in \mathcal{X}$ with $G_1 \not\simeq_{1\mathrm{WL}} G_2$. Further, let C_1, C_2 be the final colorings computed by the 1-WL algorithm when applied on G_1, G_2 respectively. Due to $G_1 \not\simeq_{1\mathrm{WL}} G_2$, there exists a color $c \in \mathbb{N}$ such that $h_{G_1,C_1}(c) \neq h_{G_2,C_2}(c)$. Such that $\mathcal{B}_c \in C$ exists with $\mathcal{B}_c(G_1) \neq \mathcal{B}_c(G_2)$. □

Lemma 19 (1-WL+NN Equivalence)**.** Let $\mathcal{B}$ be a function over $\mathcal{X}$ computable by 1-WL+NN, then for every pair of graphs $G_1, G_2 \in \mathcal{X}$: if $G_1 \simeq_{1\mathrm{WL}} G_2$ than $\mathcal{B}(G_1) = \mathcal{B}(G_2)$.

Proof. Let $\mathcal{B}$ be an arbitrary function over $\mathcal{X}$ computable by 1-WL+NN, then $\mathcal{B}$ is composed as follows: $\mathcal{B}(\cdot) = \mathrm{MLP} \circ f_{\mathrm{enc}}\{\!\{\text{1-WL}(\cdot)(v) \mid v \in V(G)\}\!\}$. Further, let $G_1, G_2 \in \mathcal{X}$ be arbitrary graphs with $G_1 \simeq_{1\mathrm{WL}} G_2$, then by definition of the relation $\simeq_{1\mathrm{WL}}$ we know that $\text{1-WL}(G_1) = \text{1-WL}(G_2)$. With this the equivalence follows immediately. □

Lemma 20 (1-WL+NN Permuation Invariance)**.** Let $\mathcal{B}$ be a function over $\mathcal{X}$ computable by 1-WL+NN, then $\mathcal{B}$ is permutation invariant.

Proof. Let $G_1, G_2 \in \mathcal{X}$ be arbitrary graphs with $G_1 \simeq G_2$ and $\mathcal{B}$ an arbitrary function computable by 1-WL+NN. Since the 1-WL algorithm is sound, we know that $G_1 \simeq G_2$ implies $G_1 \simeq_{1\mathrm{WL}} G_2$. Using Lemma 19, we can therefore conclude that: $\mathcal{B}(G_1) = \mathcal{B}(G_2)$. □

Lemma 21 (1-WL+NN Composition)**.** Let $\mathcal{C}$ be a collection of functions computable by 1-WL+NN. Further, let $h_1, \dots h_n \in \mathcal{C}$ and $\mathrm{MLP}^\bullet$ an multilayer perceptron, than the function $\mathcal{B}$ composed of $\mathcal{B}(\cdot) := \mathrm{MLP}^\bullet(h_1(\cdot), \dots, h_n(\cdot))$ is also computable by 1-WL+NN.

Proof. Assume the above and let $f_1, \dots, f_n$ be the encoding functions, as well as $\mathrm{MLP}_1, \dots, \mathrm{MLP}_n$ be the multilayer perceptrons used by $h_1, \dots h_n$ respectively. The idea of this proof is, we construct an encoding function f^* that duplicates its input and applies each encoding function f_i individually, and we construct a multilayer perceptron MLP^* that takes in this output and simulates all $\mathrm{MLP}_1, \dots, \mathrm{MLP}_n$ simultaneously. Afterwards, the given $\mathrm{MLP}^\bullet$ will be applied on the concatenation of the output of all MLP_i's. See Figure 4 for a sketch of the proof idea. A complete proof can be found in the Appendix in subsection 2.1, as this proof is very technical and not that interesting.

$$\begin{tikzcd}
& & & f_1(M_G) \arrow[r, mapsto, "\mathrm{MLP}_1"] & o_1 \arrow[dr] & & \\
G \arrow[r, mapsto, "\text{1-WL}"] & M_G \arrow[r, "f^*"] & \begin{bmatrix} f_1(M_G) \\ \vdots \\ f_n(M_G) \end{bmatrix} \arrow[ur] \arrow[dr] & \vdots & \vdots & \begin{bmatrix} o_1 \\ \vdots \\ o_n \end{bmatrix} \arrow[r, mapsto, "\mathrm{MLP}^\bullet"] & O \\
& & & f_n(M_G) \arrow[r, mapsto, "\mathrm{MLP}_n"] & o_n \arrow[ur] & &
\end{tikzcd}$$

Figure 4.: Proof idea for Lemma 21, how the constructed functions f^* and MLP* will work on input G. Here we denote with M_G the multiset of colors of the nodes of G after the 1-WL algorithm has been applied. More formally $M_G := \{\!\!\{\text{1-WL}(G)(v) \mid v \in V(G)\}\!\!\}$. Further, we let o_i be the output computed by MLP$_i$ on input $f_i(M_G)$.

$\square$

Lemma 22. Let $\mathcal{C}$ be a collection of functions from $\mathcal{X}$ to $\mathbb{R}$ computable by 1-WL+NN that is 1-WL-Discriminating. Then for all $G^* \in \mathcal{X}$, there exists a function h_{G^*} from $\mathcal{X}$ to $\mathbb{R}$ computable by 1-WL+NN, such that for all $G \in \mathcal{X} : h_{G^*}(G) = 0$ if and only if $G \simeq_{\text{1WL}} G^*$.

Proof. Assume the above. For any $G_1, G_2 \in \mathcal{X}$ with $G_1 \not\simeq_{\text{1WL}} G_2$ let $h_{G_1,G_2} \in \mathcal{C}$ be the function distinguishing them, with $h_{G_1,G_2}(G_1) \neq h_{G_1,G_2}(G_2)$. We define the function $\overline{h}_{G_1,G_2}$ working over $\mathcal{X}$ as follows:

$$\begin{aligned}
\overline{h}_{G_1,G_2}(\cdot) &= |h_{G_1,G_2}(\cdot) - h_{G_1,G_2}(G_1)| \\
&= \max(h_{G_1,G_2}(\cdot) - h_{G_1,G_2}(G_1),\ 0) + \max(h_{G_1,G_2}(G_1) - h_{G_1,G_2}(\cdot),\ 0) \\
&= \text{ReLU}(h_{G_1,G_2}(\cdot) - h_{G_1,G_2}(G_1)) + \text{ReLU}(h_{G_1,G_2}(G_1) - h_{G_1,G_2}(\cdot)) \qquad (0.1)
\end{aligned}$$

Note, that in the equations above "$h_{G_1,G_2}(G_1)$" is a fixed constant and the resulting function $\overline{h}_{G_1,G_2}$ is non-negative. Let $G_1 \in \mathcal{X}$ now be fixed, we will construct the function h_{G_1} with the desired properties as follows:

$$h_{G_1}(x) = \sum_{G_2 \in \mathcal{X},\ G_1 \not\simeq_{\text{1WL}} G_2} \overline{h}_{G_1,G_2}(x). \qquad (0.2)$$

Since $\mathcal{X}$ is finite, the sum is finite and therefore well-defined. Next, we will prove that for a fixed graph $G_1 \in \mathcal{X}$, the function h_{G_1} is correct on input $G \in \mathcal{X}$:

1. If $G_1 \simeq_{\text{1WL}} G$, then for every function $\overline{h}_{G_1,G_2}$ of the sum with $G_1 \not\simeq_{\text{1WL}} G_2$, we know, using Lemma 19, that $\overline{h}_{G_1,G_2}(G)$ is equal to $\overline{h}_{G_1,G_2}(G_1)$ which is by definition 0, such that $h_{G_1}(G) = 0$.
2. If $G_1 \not\simeq_{\text{1WL}} G$, then $\overline{h}_{G_1,G}(G)$ is a summand of the overall sum, and since $\overline{h}_{G_1,G}(G) > 0$, we can conclude $h_{G_1}(G) > 0$ due to the non-negativity of each function $\overline{h}_{G_1,G_2}$.

We can encode Equation 0.1 via a multilayer perceptron where "$h_{G_1,G_2}(G_1)$" is just a constant, and consequently can encode Equation 0.2 aswell. Therefore, we can conclude with Lemma 21 that for every graph G, h_G is also 1-WL+NN computable. $\square$

Lemma 23. Let $\mathcal{C}$ be a collection of functions from $\mathcal{X}$ to $\mathbb{R}$ computable by 1-WL+NN so that for all $G^* \in \mathcal{X}$, there exists $h_{G^*} \in \mathcal{C}$ satisfying $h_{G^*}(G) = 0$ if and only if $G \simeq_{\text{1WL}} G^*$ for all $G \in \mathcal{X}$. Then for every $G^* \in \mathcal{X}$, there exists a function φ_{G^*} computable by 1-WL+NN such that for all $G \in \mathcal{X}$: $\varphi_{G^*}(G) = \mathbb{1}_{G \simeq_{\text{1WL}} G^*}$.

Proof. Assuming the above. Due to $\mathcal{X}$ being finite, we can define for every graph G^* the constant:

$$\delta_{G^*} \coloneqq \frac{1}{2} \min_{G \in \mathcal{X},\ G \not\simeq_{\text{1WL}} G^*} |h_{G^*}(G)| > 0.$$

With this constant, we can use a so-called "bump" function working from $\mathbb{R}$ to $\mathbb{R}$ that is similar to the indicator function. We define this function for parameter $a \in \mathbb{R}$ with $a > 0$ as:

$$\begin{aligned} \psi_a(x) &\coloneqq \max(\frac{x}{a} - 1,\ 0) + \max(\frac{x}{a} + 1,\ 0) - 2 \cdot \max(\frac{x}{a},\ 0) \\ &= \text{ReLU}(\frac{x}{a} - 1) + \text{ReLU}(\frac{x}{a} + 1) - 2 \cdot \text{ReLU}(\frac{x}{a}) \end{aligned} \tag{0.3}$$

The interesting property of ψ_a is that it maps every value x to 0, except when x is being drawn from the interval $(-a, a)$. In particular, it maps x to 1 if and only if x is equal to 0. See Figure A.1 in the Appendix in subsection 1.1 for a plot of the relevant part of this function with exemplary values for a.

We use these properties to define for every graph $G^* \in \mathcal{X}$ the function $\varphi_{G^*}(\cdot) \coloneqq \psi_{\delta_{G^*}}(h_{G^*}(\cdot))$. We will quickly demonstrate that this function is equal to the indicator function, for this let G^* be fixed and G, an arbitrary graph from $\mathcal{X}$, the input:

1. If $G \simeq_{\text{1WL}} G^*$, then $h_{G^*}(G) = 0$ resulting in $\varphi_{G^*}(G) = \psi_{\delta_{G^*}}(0) = 1$.
2. If $G \not\simeq_{\text{1WL}} G^*$ then $h_{G^*}(G) \neq 0$, such that $|h_{G^*}(G)| > \delta_{G^*}$ so that $h_{G^*}(G) \notin (-\delta_{G^*}, \delta_{G^*})$ resulting in $\varphi_{G^*}(G) = 0$.

Note that we can encode φ_{G^*} using Equation 0.3 via a multilayer perceptron, where δ_{G^*} is a constant. With Lemma 21 we can therefore conclude that φ_{G^*} is computable by 1-WL+NN for every graph $G^* \in \mathcal{X}$. $\square$

Lemma 24. Let $\mathcal{C}$ be a collection of functions from $\mathcal{X}$ to $\mathbb{R}$ computable by 1-WL+NN so that for all $G^* \in \mathcal{X}$, there exists $\varphi_{G^*} \in \mathcal{C}$ satisfying $\forall G \in \mathcal{X} : \varphi_{G^*}(G) = \mathbb{1}_{G \simeq_{\text{1WL}} G^*}$, then every function computable by a GNN is also computable by 1-WL+NN.

Proof. Assume the above. For any function $\mathcal{A}$ computed by an GNN that works over $\mathcal{X}$ to $\mathbb{R}$, we show that it can be decomposed as follows for any $G \in \mathcal{X}$ as input:

$$\begin{aligned} \mathcal{A}(G) &= \Big(\frac{1}{|\mathcal{X}/{\simeq_{\text{1WL}}}(G)|} \sum_{G^* \in \mathcal{X}} \mathbb{1}_{G \simeq_{\text{1WL}} G^*} \Big) \cdot \mathcal{A}(G) \\ &= \frac{1}{|\mathcal{X}/{\simeq_{\text{1WL}}}(G)|} \sum_{G^* \in \mathcal{X}} \mathcal{A}(G^*) \cdot \mathbb{1}_{G \simeq_{\text{1WL}} G^*} \\ &= \sum_{G^* \in \mathcal{X}} \frac{\mathcal{A}(G^*)}{|\mathcal{X}/{\simeq_{\text{1WL}}}(G^*)|} \cdot \varphi_{G^*}(G) \end{aligned} \tag{0.4}$$

with $\mathcal{X}/{\simeq_{\text{1WL}}}(G^*)$ we denote the set of all graphs G over $\mathcal{X}$ that are equivalent to G^* according to the $\simeq_{\text{1WL}}$ relation.

Since $\mathcal{A}$ is permutation-invariant, and GNNs are at most as good as the 1-WL algorithm in distinguishing non-isomorphic graphs, we can use the fact that for every graph $G, H \in \mathcal{X}$ with $G \simeq_{1\text{WL}} H$: $\mathcal{A}(G) = \mathcal{A}(H)$. Therefore, we can decompose $\mathcal{A}$ as stated in Equation 0.4 via a multilayer perceptron with $\frac{\mathcal{A}(G^*)}{|\mathcal{X}/\simeq_{1\text{WL}}(G^*)|}$ being constants and $\varphi_{G^*} \in \mathcal{C}$ encoding the indicator function. Combined with the Lemma 21, we can conclude that $\mathcal{A}$ is computable by 1-WL+NN. Important to note, we can only do this since $\mathcal{X}$ is finite, making the overall sum finite and the cardinality of $\mathcal{X}/\simeq_{1\text{WL}}(G^*)$ well-defined for all graphs. □

4.2. Proof of Theorem 13

In this section we will prove the converse direction. We start with Lemma 25, where we introduce an upper bound that we will use in Lemma 26 to show that there exists a collection of GNN-computable functions that is 1-WL-Discriminating. After that, we will prove a composition lemma with Lemma 27 which is similar to the one we introduced in the previous section. From this point on, the proof continues as in the previous section and concludes the property to be proved in Lemma 28.

Lemma 25. Let G be an arbitrary graph with $n := |V(G)|$ the number of nodes and $C : V(G) \to \mathbb{N}$ an arbitrary coloring of the graph G. Then the total number of possible tuples of the form:

$$(C(v),\ \{\!\{C(u) \mid u \in \mathcal{N}(v)\}\!\}),$$

for all $v \in V(G)$ can be upper bounded by:

$$n \cdot \sum_{i=0}^{n-1} \binom{n+i-1}{i}.$$

Proof. Assume the above. For the first entry of the tuple, there exist at most n different colors, since there are n nodes. For the second entry, each node $v \in V(G)$ can have between 0 and $n-1$ neighbors, such that the total number of possibilities is the sum over each cardinality of a multiset with n different colors. In the end, we soundly combine both results by multiplying both together. □

Lemma 26 (GNN 1-WL-Discriminating)**.** There exists a collection $\mathcal{C}$ of functions from $\mathcal{X}$ to $\mathbb{R}$ computable by GNNs that is 1-WL-Discriminating. Meaning for every $G_1, G_2 \in \mathcal{X}$ with $G_1 \not\simeq_{1\text{WL}} G_2$ there exists $\mathcal{A} \in C$ such that $\mathcal{A}(G_1) \neq \mathcal{A}(G_2)$.

Proof. Since $\mathcal{X}$ is finite, we define $n := \max\{|V(G)| \mid G \in \mathcal{X}\}$ to be the maximum number of nodes of a graph in $\mathcal{X}$, and $k := \max\{l_G(v) \mid v \in V(G), G \in \mathcal{X}\}$ to be the largest label of any node of a graph in $\mathcal{X}$. Using Lemma 25, we can compute an upper bound m using n for the number of distinct tuples. Note that, this bound holds true for all graphs in $\mathcal{X}$. We construct a GNN with n layers working as follows for any $v \in V(G)$:

$$\begin{aligned} &f^{(0)}(v) := l_G(v), \\ &f^{(t)}(v) := h_{m,t}(f^{(t-1)}(v),\ \{\!\{f^{(t-1)}(u) \mid u \in \mathcal{N}(v)\}\!\}), \quad 0 < t < n. \end{aligned}$$

Here $h_{m,t}$ is a function that maps the tuples injectively to an integer of the set:

$$\{i \in \mathbb{N} \mid k + (t-1) \cdot m + 1 \leq i \leq k + t \cdot m\}$$

such that each layer uses its own coloring domain. This function exists as by the soundness of the upper bound of Lemma 25 the cardinality of its co-domain is greater or equal than the one of its domain. With this, we ensure that each layer, maps a tuple to a new, previously not used, color. Therefore, every layer of this GNN, computes a single iteration of the 1-WL algorithm. Further, since the 1-WL algorithm converges after at most $|V(G)| =: n$ iterations, we set the number of layers to n. Thereby we ensure that the coloring computed after n layers when applied on any graph $G \in \mathcal{X}$ is similarly expressive as the coloring computed by the 1-WL algorithm when applied on G.

We define the collection C of functions computable by GNNs that is 1-WL-Discriminating as:

$$C := \{\mathcal{A} : \mathcal{X} \to \mathbb{R},\ G \mapsto \mathsf{Readout}_c(\{\!\{f^{(n)}(v) \mid v \in V(G)\}\!\}) \mid c \in \mathbb{N}\},$$

where $\mathsf{Readout}_c$ is the $\mathsf{READOUT}$ function that returns the number of nodes colored as c in the coloring of $f^{(n)}$. □

Similar to the proof in the previous section, we will use Lemma 27 to introduce the ability to construct GNNs that take in as input multiple GNNs and then apply a multilayer perceptron to the combined output.

Lemma 27 (Composition GNN)**.** Let C be a collection of function computable by GNNs. Further, let $\mathcal{A}_1, \ldots, \mathcal{A}_n \in C$ and MLP a suitable multilayer perceptron, then $\hat{\mathcal{A}}(\cdot) := \mathrm{MLP}(\mathcal{A}_1(\cdot), \ldots, \mathcal{A}_n(\cdot))$ is also computable by a GNN.

Proof. Assume the above. Further, for the ease of readability we make the assumptions that the node features computed in each layer by each $\mathcal{A}_i$ is one dimensional. Note that, one can easily extend this proof to work over arbitrary feature dimensions.

Further, we will for any $x \in \mathbb{R}^d$ use the notation $x[i]$ to indicate the i.th element of the vector x. Also, we use the following notation to indicate the merge and aggregation function used in layer t by $\mathcal{A}_i$ as $f^{(t)}_{\text{merge},i}$ and $f^{(t)}_{\text{agg},i}$. Similarly, does $\mathsf{Readout}_i$ indicate the $\mathsf{READOUT}$ function and $f^{(0)}_i$ the input function of $\mathcal{A}_i$.

We will prove the lemma by giving a construction of $\hat{\mathcal{A}}$. Let K be the maximum number of layers of all $\mathcal{A}_1, \ldots, \mathcal{A}_n$. We construct the GNN $\hat{\mathcal{A}}$ with K layers, with the input layer working as follows on $v \in G$:

$$\hat{f}^{(0)}(v) := \begin{bmatrix} f^{(0)}_1(v) \\ \vdots \\ f^{(0)}_n(v) \end{bmatrix},$$

and each other layer $0 < t \leq K$ uses the merge $\hat{f}^{(t)}_{\text{merge}}$ and aggregation $\hat{f}^{(t)}_{\text{agg}}$ functions as defined below:

$$\hat{f}^{(t)}_{\text{merge}}(\hat{f}^{(t-1)}(v),\ Agg) := \begin{bmatrix} f^{(t)}_{\text{merge},i}(\hat{f}^{(t-1)}(v)[1],\ Agg[1]) \\ \vdots \\ f^{(t)}_{\text{merge},n}(\hat{f}^{(t-1)}(v)[n],\ Agg[n]) \end{bmatrix}, \text{ and}$$

$$\hat{f}^{(t)}_{\text{agg}}(\{\!\{\hat{f}^{(t-1)}(w) \mid w \in \mathcal{N}(v)\}\!\}) := \begin{bmatrix} f^{(t)}_{\text{agg},1}(\{\!\{f^{(t-1)}(w)[1] \mid w \in \mathcal{N}(v)\}\!\}) \\ \vdots \\ f^{(t)}_{\text{agg},n}(\{\!\{f^{(t-1)}(w)[n] \mid w \in \mathcal{N}(v)\}\!\}) \end{bmatrix}.$$

Note that, not all $\mathcal{A}_i$ will have K layers. For these cases we define the missing functions as follows:

$$f^{(t)}_{\text{merge},i}(\hat{f}^{(t-1)}(v),\ Agg) := \hat{f}^{(t-1)}(v), \text{ and}$$
$$f^{(t)}_{\text{agg},i}(\{\!\{f^{(t-1)}(w) \mid w \in \mathcal{N}(v)\}\!\}) := 0.$$

These functions do not change anything, and only forward the result of the actual computation of $\mathcal{A}_i$ to the last layer. Finally, we construct the $\mathsf{READOUT}$ function of $\hat{\mathcal{A}}$ as follows:

$$\mathsf{Readout}(\{\!\{\hat{f}^{(t)}(v) \mid v \in V(G)\}\!\}) := \text{MLP} \circ \begin{bmatrix} \mathsf{Readout}_i(\{\!\{f^{(t)}_{\text{merge},1}(v) \mid v \in V(G)\}\!\}) \\ \vdots \\ \mathsf{Readout}_n(\{\!\{f^{(t)}_{\text{merge},n}(v) \mid v \in V(G)\}\!\}) \end{bmatrix}.$$

□

As a result, we are in the same starting position as at the beginning of the proof in subsection 4.1. That is, we have shown with Lemma 26 that there is a collection of functions that can distinguish any pair of graphs distinguishable by the 1-WL algorithm, and with Lemma 27 we have shown that the composition of multiple GNNs and a multilayer perceptron is still computable by a GNN. Such that we can simply apply the results of Lemmas 22 and 23 to GNNs aswell and thereby concluding that for any fixed $G^* \in \mathcal{X}$ the indicator function φ_{G^*} working over $\mathcal{X}$ with:

$$\forall G \in \mathcal{X} : \varphi_{G^*}(G) := \begin{cases} 1, & \text{if } G \simeq_{1\text{WL}} G^* \\ 0, & \text{else} \end{cases},$$

is computable by a GNN.

Lemma 28. Let $\mathcal{C}$ be a collection of functions from $\mathcal{X}$ to $\mathbb{R}$ computable by GNNs so that for all $G^* \in \mathcal{X}$, there exists $\varphi_{G^*} \in \mathcal{C}$ satisfying $\forall G \in \mathcal{X} : \varphi_{G^*}(G) = \mathbb{1}_{G \simeq_{1\text{WL}} G^*}$, then every function computable by a 1-WL+NN is also computable by a GNN.

Proof. Assume the above. For any function $\mathcal{B}$ computed by 1-WL+NN that works over $\mathcal{X}$ to $\mathbb{R}$, we show that it can be decomposed as follows for any $G \in \mathcal{X}$ as input:

$$\begin{aligned} \mathcal{B}(G) &= \Big(\frac{1}{|\mathcal{X}/{\simeq_{1\text{WL}}}(G)|} \sum_{G^* \in \mathcal{X}} \mathbb{1}_{G \simeq_{1\text{WL}} G^*} \Big) \cdot \mathcal{B}(G) \\ &= \frac{1}{|\mathcal{X}/{\simeq_{1\text{WL}}}(G)|} \sum_{G^* \in \mathcal{X}} \mathcal{B}(G^*) \cdot \mathbb{1}_{G \simeq_{1\text{WL}} G^*} \\ &= \sum_{G^* \in \mathcal{X}} \frac{\mathcal{B}(G^*)}{|\mathcal{X}/{\simeq_{1\text{WL}}}(G^*)|} \cdot \varphi_{G^*}(G) \end{aligned} \tag{0.5}$$

with $\mathcal{X}/{\simeq_{1\text{WL}}}(G^*)$ we denote the set of all graphs G over $\mathcal{X}$ that are equivalent to G^* according to the $\simeq_{1\text{WL}}$ relation. With Lemma 19, we know that for any $G_1, G_2 \in \mathcal{X}$ with $G_1 \simeq_{1\text{WL}} G_2 :$ $\mathcal{B}(G_1) = \mathcal{B}(G_2)$, such this decomposition is sound.

We can encode $\mathcal{B}$ as stated in Equation 0.5 via a multilayer perceptron with $\frac{\mathcal{B}(G^*)}{|\mathcal{X}/{\simeq_{1\text{WL}}}(G^*)|}$ being constants and $\varphi_{G^*} \in \mathcal{C}$ encoding the indicator function. Combined with the Lemma 27, we can conclude that $\mathcal{B}$ is computable by 1-WL+NN. Important to note, we can only do this since $\mathcal{X}$ is finite, making the overall sum finite and the cardinality of $\mathcal{X}/{\simeq_{1\text{WL}}}(G^*)$ well-defined for all graphs. □

Part II.

Empirical Testing

Give a short Outline

5. Setup

Introduction to this section!

5.1. Choice of Datasets

Luis' Taxonomy, verschiedene Bereiche -> TUDataset

5.2. Choice of Models

GIN weil sehr expressive Xu et al. [2019], GAT und GCN wegen Nikolentzos et al. [2023] mit basic pool um complexity der models zu containen und bessere vergleiche zu haben

1-WL+NN auch nur mit basic pool

5.3. Experimental Setup

python 3.10, pytorch und pytorch-geometric (use footnotes from outline). Use of Wheights and Biases as data logging instance, HPC, Colab und meine eigene Hardware - > summierte overall computation time. Standard procedure, training with 10-fold and repition and a split of training, validation and test dataset.

6. Results

Introduction to this section!

6.1. Explain how results look like

Shortly explain, their performance and how they typically learn. Grafiken über das mean train_acc, val_acc, test_acc vs. epochs.

1-WL+NN

Explain most important parameters such as k-wl mit der wl accuracy on. Show how it relates to the datasets by showing the table of theoretical accuracies, Grafiken über das mean train_acc, val_acc, test_acc vs. epochs.

GNN

Show performance of different GNN models -> Boxplot of different pooling functions Grafiken über das mean train_acc, val_acc, test_acc vs. epochs.

6.2. Test Accuracy

Show a big table with all test-accuracies and its standard deviaton and highlight in bold face, the best one. Asses the std of both model types.

6.3. Overfitting Behaviour

Compare the difference between training data performance vs test accuracy accross both models

6.4. Aggregate Analysis

Explain what aggregates are and show tSNE with the of the best models with SVM
Also use KNN to assess how good the actuall aggregates are differentiable. -> local clusters ?

7. Discusssion

7.1. Learned Lessons

7.2. Future Work

8. Conclusion

Bibliography

[1] R. Abboud, I. I. Ceylan, M. Grohe, and T. Lukasiewicz. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

[2] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

[3] L. Babai. Lectures on graph isomorphism. University of Toronto, Department of Computer Science. Mimeographed lecture notes, October 1979, 1979.

[4] L. Babai. Graph isomorphism in quasipolynomial time. In *ACM SIGACT Symposium on Theory of Computing*, pages 684–697, 2016.

[5] L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *Symposium on Foundations of Computer Science*, pages 39–46, 1979.

[6] D. Beaini, S. Passaro, V. Létourneau, W. Hamilton, G. Corso, and P. Liò. Directional graph networks. In *International Conference on Machine Learning*, pages 748–758. PMLR, 2021.

[7] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[8] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[9] J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992.

[10] A. Cardon and M. Crochemore. Partitioning a graph in $O(|A| \log_2 |V|)$. *Theoretical Computer Science*, 19(1):85 – 98, 1982.

[11] Z. Chen, S. Villar, L. Chen, and J. Bruna. On the equivalence between graph isomorphism testing and function approximation with GNNs. In *Advances in Neural Information Processing Systems*, pages 15868–15876, 2019.

[12] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

[13] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

[14] F. Geerts. The expressive power of kth-order invariant graph networks, 2020.

[15] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.

[16] M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017.

[17] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.

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

[19] N. Immerman and E. Lander. *Describing Graphs: A First-Order Approach to Graph Canonization*, pages 59–81. Springer, 1990.

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

[21] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[22] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[23] A. Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[24] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence*, pages 4602–4609, 2019.

[25] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. *CoRR*, 2020.

[26] G. Nikolentzos, M. Chatzianastasis, and M. Vazirgiannis. What do gnns actually learn? towards understanding their representations. *arXiv preprint arXiv:2304.10851*, 2023.

[27] G. Nikolentzos, M. Chatzianastasis, and M. Vazirgiannis. Weisfeiler and leman go hyperbolic: Learning distance preserving node representations. In *International Conference on Artificial Intelligence and Statistics*, pages 1037–1054. PMLR, 2023.

[28] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[29] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[30] R. Sato, M. Yamada, and H. Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM, 2021.

[31] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[32] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

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

[34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[35] C. Vignac, A. Loukas, and P. Frossard. Building powerful and equivariant graph neural networks with structural message-passing. *Advances in neural information processing systems*, 33:14143–14155, 2020.

[36] B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

[37] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[38] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[39] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

A. Appendix Part I

1. Figures and graphs

1.1. The “Bump” Function

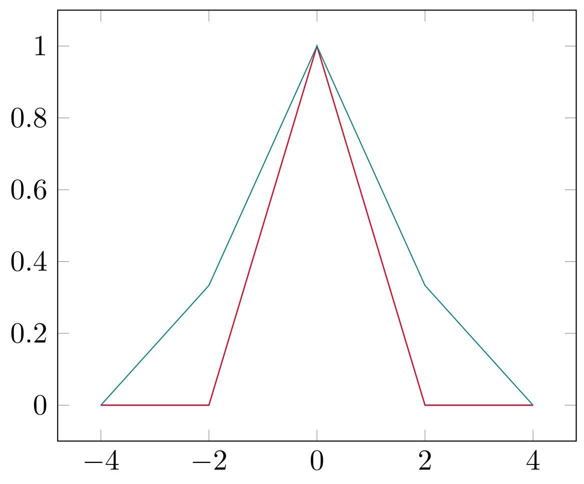


Figure A.1.: Illustration of the so-called “bump” function $\psi_a(x)$ used in the proof of Lemma 23. Here the colors of the displayed functions correspond to the parameter a set to $a := 1$ in blue, $a := 2$ in red and $a := 3$ in green.

2. Proofs

2.1. Lemma 21: Composition Lemma for 1-WL+NN

Proof of Lemma 21. Let $\mathcal{C}$ be a collection of functions computed by 1-WL+NN, $h_1, \ldots, h_n \in \mathcal{C}$, and MLP$^\bullet$ a multilayer perceptron. Further, let $f_1, \ldots, f_n$ be the encoding functions, as well as $\text{MLP}_1, \ldots, \text{MLP}_n$ be the multilayer perceptrons used by $h_1, \ldots h_n$ respectively. As outlined above, we will now construct f^* and MLP*, such that for all graphs $G \in \mathcal{X}$:

$$\text{MLP}^\bullet(h_1(G), \ldots, h_n(G)) = \text{MLP}^* \circ f^*(\{\!\{\text{1-WL}(G)(v) \mid v \in V(G)\}\!\})$$

with which we can conclude that the composition of multiple functions computable by 1-WL+NN, is in fact also 1-WL+NN computable.

We define the new encoding function f^* to work as follows on an arbitrary input multiset M:

$$f^*(M) := \mathsf{concat}(\begin{bmatrix} f_1(M) \\ \vdots \\ f_n(M) \end{bmatrix}),$$

where conccat is the concatenation function, concatenating all encoding vectors to one single vector.

Using the decomposition introduced in Definition 8, we can decompose each MLP_i for $i \in [n]$ at layer $j > 0$ as follows: $(\text{MLP}_i)_j(v) := \sigma_{i,t}(W^i_j \cdot (\text{MLP}_i)_{j-1}(v) + b^i_j)$. Using this notation we construct MLP^* as follows:

$$\begin{aligned}
&(\text{MLP}^*)_0(v) := v \\
&(\text{MLP}^*)_{j+1}(v) := \sigma^*_j(W^*_j \cdot (\text{MLP}^*)_j(v) + \mathsf{concact}(\begin{bmatrix} b^1_j \\ \vdots \\ b^n_j \end{bmatrix})) && ,\forall j \in [k] \\
&(\text{MLP}^*)_{j+k+1}(v) := (\text{MLP}^\bullet)_{j+1}(v) && ,\forall j \in [k^\bullet - 1]
\end{aligned}$$

where k is the maximum number of layers of the set of MLP_i's, and $k^\bullet$ is the number of layers of the given $\text{MLP}^\bullet$. Thereby, we define in the first equation line, that the start of the sequence is the input, with the second line, we construct the "simultaneous" execution of the MLP_i's, and in the last equation line, we add the layers of the given $\text{MLP}^\bullet$ to the end. Further, we define the weight matrix W^*_j as follows:

$$W^*_j := \begin{bmatrix} W^1_j & 0 & \dots & 0 \\ 0 & W^2_j & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & W^n_j \end{bmatrix},$$

such that we build a new matrix where each individual weight matrix is placed along the diagonal. Here we denote with "0" zero matrices with the correct dimensions, such that W^*_j is well-defined. Important to note, should for an MLP_i, W^i_j not exist, because it has less than j layers, we use for W^i_j the identity matrix I_m where m is the dimension of the output computed by MLP_i. And finally, we define the overall activation function σ^*_j as following:

$$\sigma^*_j(v) := \begin{bmatrix} \sigma_{1,j}(v[1]) \\ \vdots \\ \sigma_{1,j}(v[d_1]) \\ \vdots \\ \sigma_{n,j}(v[d_1 + \cdots + d_{n-1} + 1]) \\ \vdots \\ \sigma_{n,j}(v[d_1 + \cdots + d_n]) \end{bmatrix},$$

where d_i is the dimension of the output of MLP_i at layer j, and for the ease of readability we denote the i.th component of vector v here with $v[i]$. Thereby, we construct an activation function that applies each respective activation function of the MLP_i's individually to their respective computation. □

2.2. Theorem 15: Continuous Case: "GNN ⊆ 1-WL+NN"

We will prove Theorem 15 by first introducing a definition and then 2 lemmas using that definition which collectively prove the entire theorem. In particular, we define the property that a collection of functions is able to find any $\simeq_{1\text{WL}}$ equivalence class. Then, in Lemma 30, we prove that 1-WL-Discriminating somehow implies being able to locate any $\simeq_{1\text{WL}}$-equivalence class. And finally, in Lemma 31, we prove that being able to locate any $\simeq_{1\text{WL}}$ equivalence class further implies being somehow GNN-approximating. The overall proof is inspired by the work of [11].

Definition 29 (Locating $\simeq_{1\text{WL}}$ equivalence classes)**.** Let $\mathcal{C}$ be a collection of continuous functions from $\mathcal{X}$ to $\mathbb{R}$. If for all $\epsilon \in \mathbb{R}$ with $\epsilon > 0$ and for every graph $G^* \in \mathcal{X}$ the function h_{G^*} exists in $\mathcal{C}$, with the following properties:

1. for all $G \in \mathcal{X} : h_{G^*}(G) \geq 0$,
2. for all $G \in \mathcal{X}$ with $G \simeq_{1\text{WL}} G^* : h_{G^*}(G) = 0$, and
3. there exists a constant $\delta_{G^*} > 0$, such that for all $G \in \mathcal{X}$ with $h_{G^*}(G) < \delta_{G^*}$ there exists a graph $G' \in \mathcal{X}/{\simeq_{1\text{WL}}}(G)$ in the equivalence class of G such that $\|G' - G^*\|_2 < \epsilon$

we say $\mathcal{C}$ is able to locate every $\simeq_{1\text{WL}}$ equivalence class.

One can interpret this function h_{G^*} as a kind of loss function that measures the similarity between its input graph to G^*. It yields no loss for the input G, if G is indistinguishable from G^* by the 1-WL algorithm ($G \simeq_{1\text{WL}} G^*$), only a small loss if G is close to a graph in the $\simeq_{1\text{WL}}$ equivalence class of G^* (the loss is upper bounded by δ_{G^*}), and an arbitrary loss otherwise.

Lemma 30. Let $\mathcal{C}$ be a collection of continuous functions from $\mathcal{X}$ to $\mathbb{R}$ computable by 1-WL+NN. If $\mathcal{C}$ is 1-WL-Discriminating, then there exists a collection of functions $\mathcal{C}'$ computable by 1-WL+NN that is able to locate every $\simeq_{1\text{WL}}$ equivalence class on $\mathcal{X}$.

Proof. Let $G^* \in \mathcal{X}$ be fixed and $\epsilon > 0$ be given. Since $\mathcal{C}$ is 1-WL-Discriminating, we know that for every $G \in \mathcal{X}$ with $G \not\simeq_{1\text{WL}} G^*$, there exists a function $h_{G,G^*} \in \mathcal{C}$ such that $h_{G,G^*}(G) \neq h_{G,G^*}(G^*)$. We use this property to construct for each $G \in \mathcal{X}$ with $G \not\simeq_{1\text{WL}} G^*$ a set A_{G,G^*} as follows:

$$A_{G,G^*} \coloneqq \{G' \in \mathcal{X} \mid h_{G,G^*}(G') \in (h_{G,G^*}(G) \pm \frac{|h_{G,G^*}(G) - h_{G,G^*}(G^*)|}{2})\},$$

where $(a \pm b)$ is the open set $(a - b, a + b)$ over $\mathbb{R}$. One can use Figure A.2 below for a better understanding, when a graph G' is contained in A_{G,G^*} and when not. With the illustration one can easily see, that $G \in A_{G,G^*}$ and $G^* \notin A_{G,G^*}$.

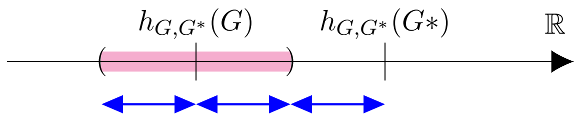


Figure A.2.: An illustration to better understand the proof. The set A_{G,G^*} consists of all graphs G that are mapped by h_{G,G^*} into the pink area, which size depends on the distance between the image of G and G^* under h_{G,G^*}. Moreover, the blue distances all have the same length.

Furthermore, by assumption h_{G,G^*} is continuous, such that A_{G,G^*} is an open set. For every $G \in \mathcal{X}$ with $G \simeq_{1\text{WL}} G^*$ we define A_{G,G^*} as follows:

$$A_{G,G^*} \coloneqq \{G' \in \mathcal{X} \mid \|G' - G\|_2 < \epsilon\},$$

where $\|\cdot\|_2$ is the l_2.

Thus, $\{A_{G,G^*}\}_{G\in\mathcal{X}}$ is an open cover of $\mathcal{X}$. Since $\mathcal{X}$ is compact, there exists a finite subset $\mathcal{X}_0$ such that $\{A_{G,G^*}\}_{G\in\mathcal{X}_0}$ also covers $\mathcal{X}$. Hence, $\forall G \in \mathcal{X} \exists G_0 \in \mathcal{X}_0 : G \in A_{G_0,G^*}$.

We define the desired function h_{G^*} as follows:

$$h_{G^*}(\cdot) = \sum_{\substack{G_0\in\mathcal{X}_0 \\ G_0 \not\simeq_{1\text{WL}} G^*}} \overline{h}_{G_0,G^*}(\cdot), \tag{A.1}$$

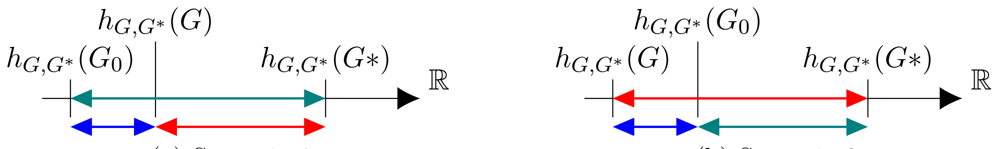
where we define $\overline{h}_{G_0,G^*}$ almost exactly the same as in a previous proof:

$$\begin{aligned} \overline{h}_{G_0,G^*}(\cdot) &= |h_{G_0,G^*}(\cdot) - h_{G_0,G^*}(G^*)| && \text{(A.2)} \\ &= \max(h_{G_0,G^*}(\cdot) - h_{G_0,G^*}(G^*)) + \max(h_{G_0,G^*}(G^*) - h_{G_0,G^*}(\cdot)) && \text{(A.3)} \end{aligned}$$

Note that, "$h_{G_0,G^*}(G^*)$" is a constant in the definition of $\overline{h}_{G_0,G^*}(\cdot)$. We will shortly proof, that $h_{G^*}(\cdot)$ fulfills the desired properties on input $G \in \mathcal{X}$:

1. By construction, any $\overline{h}_{G_0,G^*}$ is non-negative, such that the sum over these functions is also non-negative.
2. If $G \simeq_{1\text{WL}} G^*$, using Lemma 19 we know that $h_{G^*}(G) = h_{G^*}(G^*)$, and by definition $h_{G^*}(G^*) = 0$, such that we can conclude $h_{G^*}(G) = 0$.
3. Let δ_{G^*} be:
$$\delta_{G^*} := \frac{1}{2} \min_{\substack{G_0\in\mathcal{X} \\ G_0 \not\simeq_{1\text{WL}} G^*}} |h_{G_0,G^*}(G_0) - h_{G_0,G^*}(G^*)|.$$
Prove by contraposition: Assume that for every graph $G' \in \mathcal{X}/\simeq_{1\text{WL}}(G)$: $\|G' - G^*\|_2 \geq \epsilon$. Hence, $G \notin \bigcup_{G'\in\mathcal{X}/\simeq_{1\text{WL}}(G^*)} A_{G',G^*}$ (not in the union of l_2 balls of size ϵ around all graphs of the equivalence class of G^*). However, since $\{A_{G,G^*}\}_{G\in\mathcal{X}_0}$ is a cover of $\mathcal{X}$, there must exist a $G_0 \in \mathcal{X}_0$ with $G_0 \not\simeq_{1\text{WL}} G^*$ such that $G \in A_{G_0,G^*}$. Thus, by definition of A_{G_0,G^*} we know that $h_{G_0,G^*}(G) \in (h_{G_0,G^*}(G_0) \pm \frac{|h_{G_0,G^*}(G_0) - h_{G_0,G^*}(G^*)|}{2})$, which when reformulated states:
$$|h_{G_0,G^*}(G) - h_{G_0,G^*}(G_0)| < \frac{1}{2}|h_{G_0,G_0}(G) - h_{G_0,G^*}(G^*)|. \tag{A.4}$$
Using this, we can prove $\overline{h}_{G_0,G^*}(G) \geq \delta_{G^*}$, which implies $h_{G^*}(G) \geq \delta_{G^*}$ and concludes the proof:
$$\begin{aligned} \overline{h}_{G_0,G^*}(G) &= |h_{G_0,G^*}(G) - h_{G_0,G^*}(G^*)| \\ &\geq |h_{G_0,G^*}(G_0) - h_{G_0,G^*}(G^*)| - |h_{G_0,G^*}(G) - h_{G_0,G^*}(G_0)| && \text{(A.5)} \\ &\geq \frac{1}{2}|h_{G_0,G^*}(G_0) - h_{G_0,G^*}(G^*)| && \text{(A.6)} \\ &\geq \frac{1}{2} \min_{\substack{G_0\in\mathcal{X} \\ G_0 \not\simeq_{1\text{WL}} G^*}} |h_{G_0,G^*}(G_0) - h_{G_0,G^*}(G^*)| =: \delta_{G^*} \end{aligned}$$
To understand these inequalities, it helps to visualize them, see therefore Figure A.3. We will try to give an explanation now, using the colored distances depicted in Figure A.3. In Equation A.5, we use the fact that the red distance is always greater than the green minus the blue distance. Further, using Equation A.4, we know that the blue distance is always smaller than half of than the green distance. Using this fact, it is easy to see that

in Equation A.6 the green minus the blue distance is always greater than or equal to the half of the green distance.



(a) Scenario 1 (b) Scenario 2

Figure A.3.: An illustration of two general scenarios to better understand the above used inequalities. Note that, there exists two more general scenarios, with $h_{G_0,G^*}(G^*) < h_{G_0,G^*}(G_0)$, but since we use the absolute function as our measure of distance, these scenarios are equivalent to the ones depicted due to the symmetry of the absolute function. In Table A.1 below, we listed all colors used to decode the distances in the figures with their corresponding term in the inequalities.

Color	Term
red	$\lvert h_{G_0,G^*}(G) - h_{G_0,G^*}(G^*)\rvert$
green	$\lvert h_{G_0,G^*}(G_0) - h_{G_0,G^*}(G^*)\rvert$
blue	$\lvert h_{G_0,G^*}(G) - h_{G_0,G^*}(G_0)\rvert$

Table A.1.: Color legend for the proof of Lemma 30

Consequently, we know that if $h_{G^*}(G) < \delta_G$ it means that $G \in \bigcup_{G' \in \mathcal{X}/\simeq_{\text{1WL}}(G^*)} A_{G',G^*}$, which implies that there exists $G' \in \mathcal{X}/\simeq_{\text{1WL}}(G)$ with $\|G' - G^*\|_2 < \epsilon$.

As a final note, we can construct a multilayer perceptron $\text{MLP}_{h_{G^*}}$ computing the function $h_{G^*}(\cdot)$ as in Equation A.1. The $\text{MLP}_{h_{G^*}}$ gets as input a vector of the output of the finite set of functions $\{\overline{h}_{G_0,G^*}\}_{G_0 \in \mathcal{X}_0,\ G_0 \not\simeq_{\text{1WL}} G^*}$ applied on the input graph. Further, Equation A.1 can be encoded by replacing the max operator by the non-linear activation function ReLU. Using Lemma 21, we can conclude that $h_{G^*}(\cdot)$ is computable by 1-WL+NN.

□

Lemma 31. Let $\mathcal{C}$ be a collection of continuous functions from $\mathcal{X}$ to $\mathbb{R}$. If $\mathcal{C}$ is able to locate every $\simeq_{\text{1WL}}$ equivalence class on $\mathcal{X}$, then there exists a collection of functions $\mathcal{C}'$ computable by 1-WL+NN that is able to locate every $\simeq_{\text{1WL}}$ equivalence class on $\mathcal{X}$.

Proof. Let $\mathcal{A}$ be a continuous function from $\mathcal{X}$ to $\mathbb{R}$ computable by a GNN. Since $\mathcal{X}$ is compact, this implies that $\mathcal{A}$ is uniformaly continuous on $\mathcal{X}$, which further implies that $\forall \epsilon > 0 \exists r > 0$ such that $\forall G_1, G_2 \in \mathcal{X}$, if $\|G_1 - G_2\|_2 < r$, then $|\mathcal{A}(G_1) - \mathcal{A}(G_2)| < \epsilon$. Further, since $\mathcal{A}$ is GNN computable, we know that $\forall G_1, G_2 \in \mathcal{X}$, if $G_1 \simeq_{\text{1WL}} G_2 : \mathcal{A}(G_1) = \mathcal{A}(G_2)$, hence if $G' \in \mathcal{X}/\simeq_{\text{1WL}}(G_1)$ with $\|G' - G_2\|_2 < r$ exists, than $|\mathcal{A}(G_1) - \mathcal{A}(G_2)| < \epsilon$.

Throughout the rest of the proof, let $\epsilon > 0$ be fixed. Using the property described above, we know that for this ϵ there exists a constant r, we fix this aswell. Further, by the assumptions of the lemma we try to prove, we know that for any $\epsilon' > 0$ there exists $h_G \in \mathcal{C}$ for any $G \in \mathcal{X}$ with the above described properties. We choose $\epsilon' := r$ for all $h_G \in \mathcal{C}$ throughout the proof.

For any $G \in \mathcal{X}$, we define the set $h_G^{-1}(a) := \{G' \in \mathcal{X} \mid h_G(G') \in [0,a)\}$, as the set of graphs that are mapped into the open interval $[0,a)$ by h_G. We illustrated this set in Figure A.4 for better understanding.

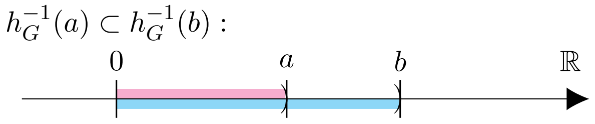


Figure A.4.: Illustration of the set $h_G^{-1}(\cdot)$ for two arbitrary values a, b with $a < b$. The pink area visualizes the open interval $[0,a)$, and similarly in blue for $[0,b)$.

Then, by definition of h_G, there exists a constant δ_G such that:

$$h_G^{-1}(\delta_G) \subseteq \bigcup_{G' \in \mathcal{X}/\simeq_{1\text{WL}}(G)} \{G'' \in \mathcal{X} \mid \|G'' - G'\|_2 < r\}.$$

Since h_G is continuous by definition, $h_G^{-1}(\delta_G)$ is an open set. Hence, $\{h_G^{-1}(\delta_G)\}_{G \in \mathcal{X}}$ is an open cover of $\mathcal{X}$, and further as $\mathcal{X}$ is compact, there exists a finite subset $\mathcal{X}_0 \subseteq \mathcal{X}$ such that $\{h_{G_0}^{-1}(\delta_{G_0})\}_{G_0 \in \mathcal{X}_0}$ is also a cover of $\mathcal{X}$. For each $G_0 \in \mathcal{X}_0$, we construct the function φ_{G_0} from $\mathcal{X}$ to $\mathbb{R}$ as follows:

$$\varphi_{G_0}(\cdot) := \max(\delta_{G_0} - h_{G_0}(\cdot),\ 0).$$

The function has two important properties, for once it is non-negative and for any $G \in \mathcal{X}$: $\varphi_{G_0}(G) > 0$, if and only if $G \in h_{G_0}^{-1}(\delta_{G_0})$, thereby acting as a sort of weak indicator function. Building up on this property, we construct for each $G_0 \in \mathcal{X}_0$ the function ψ_{G_0} from $\mathcal{X}$ to $\mathbb{R}$ as follows:

$$\psi_{G_0}(\cdot) := \frac{\varphi_{G_0}(\cdot)}{\sum_{G' \in G_0} \varphi_{G'}(\cdot)},$$

which is well-defined, because $\{h_{G_0}^{-1}(\delta_{G_0})\}_{G_0 \in \mathcal{X}_0}$ is a cover of $\mathcal{X}$, such that we can conclude that for any input graph G on $\psi_{G_0}(\cdot)$ there exists a set $h_{G_0}^{-1}(\delta_{G_0})$ with $G \in h_{G_0}^{-1}(\delta_{G_0})$ implying $\varphi_{G_0}(G) > 0$, thus making the denominator not 0. The function ψ_{G_0} has two important properties, for once it is non-negative, because φ_G for all $G \in \mathcal{X}$ is non-negative, and for any $G \in \mathcal{X}$: $\psi_{G_0}(G) > 0$, if and only if $G \in h_{G_0}^{-1}(\delta_{G_0})$.

Further, we can observe that the set of functions $\{\psi_{G_0}\}_{G_0 \in \mathcal{X}_0}$ is a partition of unity on $\mathcal{X}$ with respect to the open cover $\{h_{G_0}^{-1}(\delta_{G_0})\}_{G_0 \in \mathcal{X}_0}$, because:

1. For any $G \in \mathcal{X}$ the set of functions mapping G not to 0 is finite, as the set of all functions $\{\psi_{G_0}\}_{G_0 \in \mathcal{X}_0}$ is finite, since $\mathcal{X}_0$ is finite.
2. For any $G \in \mathcal{X}$: $\sum_{G_0 \in \mathcal{X}_0} \psi_{G_0}(G) = 1$, since:

$$\sum_{G_0 \in \mathcal{X}_0} \psi_{G_0}(G) \; = \sum_{G_0 \in \mathcal{X}_0} \frac{\varphi_{G_0}(G)}{\sum_{G' \in \mathcal{X}_0} \varphi_{G'}(G)} = \frac{\sum_{G_0 \in \mathcal{X}_0} \varphi_{G_0}(G)}{\sum_{G' \in \mathcal{X}_0} \varphi_{G'}(G)} = 1.$$

We can use this property to decompose the given function $\mathcal{A}$ as follows on any input $G \in \mathcal{X}$:

$$\mathcal{A}(G) = \mathcal{A}(G) \cdot \Big(\sum_{G_0 \in \mathcal{X}_0} \psi_{G_0}(G) \Big) = \sum_{G_0 \in \mathcal{X}_0} \mathcal{A}(G) \cdot \psi_{G_0}(G).$$

Recall the property from the beginning of the proof that for any $G \in \mathcal{X}$ if $G \in h_{G_0}^{-1}(\delta_{G_0})$, then there exists a $G' \in \mathcal{X}/\!\simeq_{1\mathrm{WL}}(G)$ with $\|G' - G_0\|_2 < r$, which implies that $|\mathcal{A}(G) - \mathcal{A}(G_0)| < \epsilon$. With this, we can construct a function $\hat{\mathcal{A}}$ on $\mathcal{X}$ that approximates $\mathcal{A}$ within accuracy ϵ:

$$\hat{\mathcal{A}}(\cdot) = \sum_{G_0 \in \mathcal{X}_0} \mathcal{A}(G_0) \cdot \psi_{G_0}(\cdot).$$

Note that, "$\mathcal{A}(G_0)$" is a constant here. To prove that $\hat{\mathcal{A}}$ approximates $\mathcal{A}$ within accuracy ϵ we need to show that $\sup_{G \in \mathcal{X}} |\mathcal{A}(G) - \hat{\mathcal{A}}(G)| < \epsilon$. Let $G \in \mathcal{X}$ be arbitrary, then:

$$\begin{aligned}
|\mathcal{A}(G) - \hat{\mathcal{A}}(G)| &= \Big| \sum_{G_0 \in \mathcal{X}_0} \mathcal{A}(G) \cdot \psi_{G_0}(G) \; - \sum_{G_0 \in \mathcal{X}_0} \mathcal{A}(G_0) \cdot \psi_{G_0}(G) \Big| \\
&= \sum_{G_0 \in \mathcal{X}_0} |\mathcal{A}(G) - \mathcal{A}(G_0)| \cdot \psi_{G_0}(G) \\
&< \sum_{G_0 \in \mathcal{X}_0} \epsilon \cdot \psi_{G_0}(G) \qquad \text{(A.7)} \\
&= \epsilon \cdot \sum_{G_0 \in \mathcal{X}_0} \psi_{G_0}(G) \\
&= \epsilon \cdot 1
\end{aligned}$$

In Equation A.7, we use the fact that applies to all $G_0 \in \mathcal{X}_0$ on input G:

- If $\psi_{G_0}(G) > 0$, than we know that $G \in h_{G_0}^{-1}(\delta_{G_0})$, such that we can upper bound $|\mathcal{A}(G) - \mathcal{A}(G_0)| < \epsilon$.
- If $\psi_{G_0}(G) = 0$, we know that the no matter what $|\mathcal{A}(G) - \mathcal{A}(G_0)|$ is, the summand is 0, such that we can just assume $|\mathcal{A}(G) - \mathcal{A}(G_0)| < \epsilon$ without loss of generality.

In the end, we give a short explanation, that $\hat{\mathcal{A}}$ is computable by 1-WL+NN. For this we construct a multilayer perceptron with three layers and then conclude with Lemma 21 the computability. Let us therefore break down the whole construction of $\hat{\mathcal{A}}$:

$$\hat{\mathcal{A}}(\cdot) = \sum_{G_0 \in \mathcal{X}_0} \mathcal{A}(G_0) \cdot \frac{1}{\sum_{G' \in \mathcal{X}_0} \max(\delta_{G'} - h_{G'}(\cdot),\ 0)} \cdot \max(\delta_{G_0} - h_{G_0}(\cdot),\ 0).$$

We construct a multilayer perceptron $\mathrm{MLP}_{\hat{\mathcal{A}}}$ that takes in as input a vector of the output of the finite set of functions $\{h_{G_0}\}_{G_0 \in \mathcal{X}_0}$ applied on the input graph. In the first layer we compute each $\max(\delta_{G_0} - h_{G_0}(\cdot))$ term, where δ_{G_0} is a constant, in particular the bias, and the max operator is replaced by the activation function ReLU. In the second layer, we compute the sum of the denominator of the fraction, to which we apply the activation function $f(x) := \frac{1}{x}$. In the last layer we compute the overall sum where $\mathcal{A}(G_0)$ is a constant. □

Theorem 16: Continuous Case: "1-WL+NN $\subseteq$ GNN"

In this section we will shortly prove that GNN-Approximating implies 1-WL-Discriminating. Similar to the proof of Theorem 12, we will take advantage of the fact that our Definition 7 of GNNs does not impose any constraints on the readout function, apart from the fact that it must be permutation invariant and computable. We have drawn inspiration for this proof from the work of [11].

Not yet adapted to the new definition of the readout functions

Proof. □

Proof of Theorem 16. Let $\mathcal{C}$ be a collection of continues functions from $\mathcal{X}$ to $\mathbb{R}$ that is GNN approximating. Let $G_1, G_2 \in \mathcal{X}$ with $G_1 \not\simeq_{1\text{WL}} G_2$, then we define the function h_{G_1,G_2} on input $G \in \mathcal{X}$:

$$h_{G_1,G_2}(G) = \min_{\pi \in S_n} \|\pi^T G \pi - G_1\|_2,$$

New proof will be very similar to the finite case

where $\|\cdot\|_2$ is the l_2 norm. Note that, h_{G_1,G_2} is computable as the number of permutations π in S_n are finite. Since h_{G_1,G_2} is permutation invariant, we can construct a GNN with 0 layers and h_{G_1,G_2} as its $\mathsf{READOUT}$ function, thereby constructing a GNN computing h_{G_1,G_2} and consequently showing that the function is GNN computable. We choose $\epsilon := \frac{1}{2} \cdot h_{G_1,G_2}(G_2) > 0$, then there exists a function $h_\epsilon \in \mathcal{C}$ approximating h_{G_1,G_2} within ϵ accuracy. With this, we have a function $h_\epsilon \in \mathcal{C}$ that can distinguish G_1 and G_2, since:

$$\begin{aligned}
|h_\epsilon(G_1) - h_\epsilon(G_2)| &> |(h_{G_1,G_2}(G_1) - \epsilon) - (h_{G_1,G_2}(G_2) + \epsilon)| \\
&= |h_{G_1,G_2}(G_2) - 2 \cdot \epsilon| && \text{by definition } h_{G_1,G_2}(G_1) = 0 \\
&= |h_{G_1,G_2}(G_2) - 2 \cdot \frac{1}{2} \cdot h_{G_1,G_2}(G_2)| = 0.
\end{aligned}$$

$\square$

B. Appendix Part II

More results