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

July 13, 2023

Acknowledgements

Luis, Christopher und HPC RWTH

Contents

1. Background and Related Work

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

In der ganzen Section passiert nichts dramatisches. Wäre super wenn du einmal einfach kurz drüber lesen kannst, und schaust das kein gravierender Fehler irgendwo steckt. Wäre nur super zu wissen ob der Lesefluss gut ist und der Leser versteht um was es hier geht in der Arbeit.

1.1. 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. each color appears equally often in the set of the nodes direct neighbors. 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 Section 2.3.

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][1]. 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]).

1.2. Graph Neural Networks

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

[1] 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.

past decades. However, researchers faced challenges in effectively adapting these techniques to graphs of diverse sizes and complexities in the early stages. 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 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 GNN models, 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. The computation of each new node feature involves aggregating all the features of the neighboring nodes and the node's own feature. After applying each layer of a GNN model, a representation of the entire graph is obtained by applying a pooling function (e.g. Ying et al. [2018]). 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 in Section 2.5; 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 GNNs 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] has 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. [2023b], 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 GNN models and not be generalized. In another publication, Nikolentzos et al. [2023a] presented mathematical proof and empirical confirmation showing how much structural information is encoded by modern GNN models. Their 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 layers of message-passing with features that relate with the number of walks of length k over the input graph form each node, disregarding the local structure within the nodes are contained.

Part I.

Theoretical Equivalence

This part of the thesis focuses on the equivalence between 1-WL+NN and GNN. We will begin by providing a preliminary section that formalizes all the concepts used in the proof and introduces a general notation. Afterward, we will dedicate a separate section to present and prove three theorems. These theorems combined conclude the equivalence.

2. Preliminaries

This section will introduce and formalizes all concepts used throughout the proof and the rest of the thesis. 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.

In der ganzen Section passiert nichts dramatisches. Wäre super wenn du einmal einfach kurz drüber lesen kannst, und schaust das kein gravierender Fehler irgendwo steckt. Nur die Definitionen von 1-WL+NN und GNN sind wichtig hier, der Rest ist eigentlich sehr normal.

2.1. 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 $\{0, \ldots, n\} \subset \mathbb{N}$ 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.

2.2. Graphs

We will briefly introduce a formal definition for graphs and coloring on graphs. Starting with the definition of a graph.

Definition 1 (Graph)**.** A graph G is defined as a 3-tuple denoted by $G := (V, E, l)$. This tuple consists of a set of nodes $V \subset \mathbb{N}$, a set of edges $E \subseteq V \times V$, and a labeling function $l : M \to \Sigma$. The domain M of the labeling function can be either V, $V \cup E$, or E, and the codomain Σ is an alphabet with $\Sigma \subseteq \mathbb{N}^k$, where $k \in \mathbb{N}$ is arbitrary. In the context of this thesis, the assigned values by the labeling function are referred to as either labels or features, depending on the dimension of Σ. In detail, if $k = 1$, we usually refer to the values as labels, otherwise as features. Additionally, the set of all graphs is denoted by $\mathcal{G}$.

Furthermore, a graph G can be either directed or undirected based on the definition of its set of edges E. If $E \subseteq \{(v, u) \mid v, u \in V\}$, it represents a directed graph, whereas if $E \subseteq \{(v, u) \mid v, u \in V, v \neq u\}$ such that for every $(v, u) \in E$ there exists $(u, v) \in E$, it defines an undirected graph. Additionally, for ease of notation, we will use $V(G)$ and $E(G)$ to denote the set of nodes and the set of edges of G, respectively, as well as l_G to denote the label function of G. Further, with $\mathcal{N}(v)$ for $v \in V(G)$ we denote the set of neighbors of v defined as $\mathcal{N}(v) := \{u \mid (u, v) \in E(G)\}$, and with $d(v)$ for $v \in V(G)$ the degree of node v, defined as $d(v) := |\mathcal{N}(v)|$.

We continue with the definition of a graph coloring.

Definition 2 (Graph Coloring)**.** 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 $\mathcal{P}$ 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 $n \in \mathbb{N}$. 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 $V(G_\pi) = \{\pi(0), \ldots, \pi(n)\}$ 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 3 (Permutation Invariant)**.** Let $f : \mathcal{G} \to \mathcal{Y}$ be an arbitrary function, then f is *permutation-invariant* if and only if for all $G \in \mathcal{G}$, where $n_G :=\mid V(G) \mid$ and for every $\pi \in S_{n_G}$: $f(G) = f(\pi \cdot G)$.

Definition 4 (Permuation Equivariant)**.** Let $f : \mathcal{G} \to \mathcal{Y}$ be an arbitrary function, then f is *permuation-equivariant* if and only if for all $G \in \mathcal{G}$, where $n_G :=\mid V(G) \mid$ and for every $\pi \in S_{n_G}$: $f(G) = \pi^{-1} \cdot f(\pi \cdot G)$.

2.3. Weisfeiler and Leman Algorithm

The Weisfeiler-Leman algorithm consists of two main parts: the coloring algorithm and the graph isomorphism test. We will introduce each part individually and present some implications afterward.

The Weisfeiler-Leman Graph Coloring Algorithm

The 1-WL algorithm computes a node coloring of its input graph in each iteration. In detail, a color is assigned to each node based on the colors of its neighbors and its own current color. The algorithm continues until convergence is reached, resulting in the final coloring of the graph. We will now formally define this procedure and provide an illustrative example in Figure 1.

Definition 5 (1-WL Algorithm)**.** Let $G = (V, E, l)$ be a labeled graph. In each iteration i, the 1-WL algorithm computes a node coloring $C_i : V(G) \to \mathbb{N}$. In the initial iteration $i = 0$, the coloring is set to $C_0 = l$ if l exists. Otherwise, for all $v \in V(G) : C_0(v) = c$ with $c \in \mathbb{N}$ being an arbitrary but fixed constant. 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, color. 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 $\mathsf{1\text{-}WL}(G) := C_\infty$.

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 algorithm, see Figure 1. 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 $\mathcal{O}(|V(G)| + |E(G)| \cdot \log |V(G)|)$.

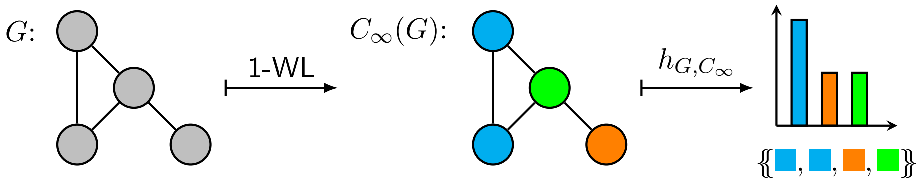


Figure 1.: An example of the final coloring computed by applying the 1-WL algorithm on the graph G. The graph G consists of 4 nodes with all their labels being set to the same color.

It is important to understand that since the algorithm was originally developed as a simple heuristic for the *graph isomorphism problem*, which is an inherently discrete problem, the 1-WL algorithm in its simplest form, as we presented it here, does only work on graphs with discrete, one-dimensional node labels. Although it is quite easy to adapt the algorithm to respect discrete edge labels of a graph by using them as weights in the neighborhood aggregation (Shervashidze et al. [2011]), modifying its definition to work with continuous graph features is more complex. Numerous proposed modifications have been put forward to address this integration in the literature, such as those discussed by Morris et al. [2016]. However, note that this particular topic will not be further investigated in this thesis, although its mention holds value for the following section.

The Weisfeiler-Leman Graph Isomorphism Test

The isomorphism test uses the 1-WL coloring algorithm and is defined as follows.

Definition 6 (1-WL Isomorphism Test)**.** To determine if two graphs $G, H \in \mathcal{G}$ are non-isomorphic ($G \ncong H$), 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:

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

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

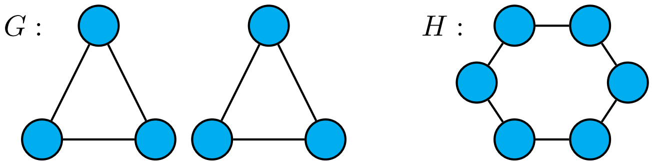


Figure 2.: This is an example of two graphs G and H that are non-isomorphic but cannot be distinguished by the 1-WL isomorphism test.

Implications of the 1-WL Algorithm

One intriguing implication of the 1-WL algorithm and its isomorphism test is that, due to it not being complete for solving the *graph isomorphism problem*, it gives rise to a related but weaker relation than the isomorphism relation ($\simeq$). We define this relation as follows.

Definition 7 (1-WL Relation)**.** Let $\mathcal{X} \subseteq \mathcal{G}$. 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.

Similarly, we define the notion 1-WL-Discriminating for collections of permutation invariant functions.

Definition 8 (1-WL-Discriminating)**.** Let $\mathcal{X} \subseteq \mathcal{G}$. Further, 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 $h(G_1) \neq h(G_2)$.

2.4. 1-WL+NN

As the previous section shows, the 1-WL algorithm is quite powerful in identifying a graph's substructures and distinguishing non-isomorphic graph pairs. With the 1-WL+NN framework, we define functions that utilize this structural information to derive application-specific insights.

Definition 9 (1-WL+NN)**.** A 1-WL+NN model consists of three components that are applied sequentially to its input: 1. the 1-WL algorithm, 2. an encoding function f_{enc} operating on graph colorings, and 3. an arbitrary multilayer perceptron MLP. In detail, a 1-WL+NN model computes the function $\mathcal{B}$, that is defined as follows:

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

where "1-WL(G)" is the coloring computed by the 1-WL algorithm when applied on G, and $k \in \mathbb{N}$ is an arbitrary constant. For a better understanding and an illustrative explanation, see Figure 3.

It is worth noting that this definition can be easily adjusted to accommodate node or edge-related tasks by applying the encoding function f_{enc} and the multilayer perceptron MLP elementwise to the colors of the multiset. However, for the purposes of this thesis, we will

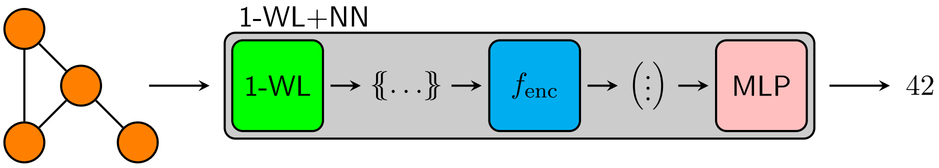


Figure 3.: This simplified illustration explains the components that make up a 1-WL+NN model and how each one processes the input further. In detail, the model takes the graph on the left as input and first applies the 1-WL algorithm, thereby obtaining a multiset of the colors assigned by the algorithm. Then the encoding function f_{enc} is applied, resulting in a fixed-sized vector that is further processed by the multilayer perceptron MLP. The output of the MLP is then propagated as the 1-WL+NN models output, here the number 42.

not delve into these variations, as our main focus will be on graph-wide tasks such as graph classification or regression, which possess greater theoretical interest and are more prevalent in most datasets. Furthermore, all the theoretical findings presented in this thesis can also be applied to 1-WL+NN models designed for node or edge tasks.

2.5. 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. Each GNN layer thus technically obtains a new graph structurally identical to the previous one but contains new feature information. After an input graph has been passed through all layers, a final readout function is applied that pools all graph features and derives a task-related 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 only to consider node features; however, one can easily extend it to include edge features as well.

Definition 10 (Graph Neural Network)**.** Let $G = (V, E, l)$ be an arbitrary graph. A GNN is a composition of multiple layers and a final readout function where each layer t is represented by a function $f^{(t)}$. The initial layer at $t = 0$ is a functioning of the format $f^{(0)} : V(G) \to \mathbb{R}^{1\times d}$ that is consistent with l and translates all labels into a vector representation. In contrast, for every $t > 0$, $f^{(t)}$ is recursively defined as:

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

where $f^{(t)}_{\text{merge}}$ is an arbitrary function that maps the aforementioned tuple to a vector, effectively “merging” them, while $f^{(t)}_{\text{agg}}$ is an arbitrary function that maps the multiset to a vector, effectively “aggregating” it.

The readout function, referred to as Readout, is applied after the input graph has been passed subsequently through all layers and is defined as follows:

$$\mathsf{Readout}(\{\!\{f^{(t)}(v) \mid v \in V(G)\}\!\}).$$

This function pools the information from every node feature, processes it, and calculates a fixed-sized output vector for the entire graph.

In summary, a GNN model will compute the function $\mathcal{A}$ defined as follows:

$$\mathcal{A} : \mathcal{G} \to \mathbb{R}^k, \; G \mapsto \mathsf{Readout}(\{\!\{ f^{(T)}(v) \mid v \in V(G) \}\!\}),$$

where T is the number of layer of the GNN, and $k \in \mathbb{N}$ an arbitrary constant. To enable end-to-end training of a GNN, it is essential that all its components are differentiable. Therefore, we require all $f^{(t)}_{\text{merge}}$ and $f^{(t)}_{\text{agg}}$ functions for all $t \in [T]$, along with the final Readout function, to be differentiable.

Note that, due to our definition of the "aggregation" and the "readout" function to operate over multisets, both functions are permutation invariant by definition. With this, we can conclude that the total composition $\mathcal{A}$ is permutation invariant, and with similar reasoning, it is also differentiable. This property 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. Furthermore, GNNs following this definition are regarded as Message-Passing-Neural-Network (MPNN). This designation stems from each node exchanging information with its direct neighbors in each layer. As a result, information during the processing of a graph is propagated by passing many messages across the graph; thus, these layers are also referred to as *message-passing* layers. As outlined in the introduction to this thesis, we will solely focus on GNNs utilizing the *message-passing* architecture. Therefore we will use the term GNN and MPNN interchangeably throughout this thesis. The definition and notation used here are inspired by Morris et al. [2019] and Xu et al. [2019].

To bridge the gap from the theoretical definition to practical instances of the definition, we will now introduce three distinct GNN architectures. Specifically, we will explore the Graph Attention Network (GAT) developed by Veličković et al. [2017], Graph Convolutional Network (GCN) proposed by Kipf and Welling [2017], and the Graph Isomorphism Network (GIN) introduced by Xu et al. [2019]. These architectures will serve as empirical baselines in Part II. Additionally, we will also elaborate on the reasons for this choice of models in Part II in **??**. We listed the definitions of the *message-passing* layers of each model in Table 1.

Table 1.: Overview of the construction of the *message-passing* layers and their respective learnable parameters by popular GNN model. This format is inspired by Nikolentzos et al. [2023a].

Model	Message-Passing Layer Definition	Learnable Parameters
GAT	$f^{(t)}(v) = \sigma\Big(\sum_{u \in \mathcal{N}(v)} \alpha_{vu} \cdot W^{(t)} \cdot f^{(t-1)}(u)\Big)$ with $\quad \alpha_{vu} = \frac{\exp\big(\text{LeakyReLU}\big(\vec{a}^T \cdot \mathsf{concat}[W^{(t)} f^{(t-1)}(v),\; W^{(t)} f^{(t-1)}(u)]\big)\big)}{\sum_{k \in \mathcal{N}(v)} \exp\big(\text{LeakyReLU}\big(\vec{a}^T \cdot \mathsf{concat}[W^{(t)} f^{(t-1)}(v),\; W^{(t)} f^{(t-1)}(k)]\big)\big)}$	$\vec{a}, W^{(t)}$
GCN	$f^{(t)}(v) = \text{ReLU}\Big(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{W^{(t)}}{\sqrt{(1+d(v)) \cdot (1+d(u))}} f^{(t-1)}(u)\Big)$	$W^{(t)}$
GIN	$f^{(t)}(v) = \mathsf{MLP}^{(t)}\Big((1 + \epsilon^{(t)}) \cdot f^{(t-1)}(v) + \sum_{u \in \mathcal{N}(v)} f^{(t-1)}(u)\Big)$	$\epsilon^{(t)}, \mathsf{MLP}^{(t)}$

Definition von Attention Matrix (GAT) ist sehr hässlich. Kann ich die einfach weglassen, und α_{vu} als "learnable Parameter" abstemplen?

Commonly employed Readout functions in this context often involve straightforward pooling techniques like elementwise summation, mean calculation, or maximum extraction. These pooling operations are typically followed by a multilayer perceptron, which performs additional processing on the aggregated information. Although more sophisticated pooling operations exist, such as SET2SET developed by Vinyals et al. [2015], Xu et al. [2019] showed that given the correct configuration, the elementwise summation pooling function combined with a following multilayer perceptron suffices to create a GNN that is as expressive as the 1-WL algorithm in distinguishing non-isomorphism.

3. Theoretical Connection

This section is the main part of our theoretical investigation of the two frameworks 1-WL+NN and GNN. We will present three intriguing theorems, which will be proven separately afterward. In detail, the first two theorems will establish an equivalence between the two frameworks when the input set of graphs is finite. While the last theorem will take this a step further and demonstrate how powerful the 1-WL algorithm is by establishing a connection between 1-WL+NN and GNN for continuous functions.

Das hier ist der wichtigste Teil. Insbesondere die 3 Beweise der Theoreme die ich hier presentiere.

In the first two theorems, we focus on a finite collection of graphs, which we denote by $\mathcal{X}$ with $\mathcal{X} \subset \mathcal{G}$.

Theorem 11 (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 12 (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 two theorems, the equivalence between both frameworks follows. Specifically, every function computed by 1-WL+NN working over any arbitrary, but finite $\mathcal{X} \subset \mathcal{G}$ is also computable by a GNN, and vice versa. As we move towards the empirical evaluation in Part II, it is evident that if we test a 1-WL+NN model on any of the benchmark datasets, it can achieve theoretically the same level of performance as a GNN model. Notice that we did not leverage any constraints on the encoding of graphs throughout the first two theorems and their corresponding proves but instead 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 the 1-WL algorithm and in particular of collections of functions that are 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, as outlined in Section 2.4, 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. Therefore, we assume in the proof and the following theorem that such a continuous version of the 1-WL algorithm exists.

We define the set of graphs with continuous labels using the following definition.

Definition 13. 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 theorem.

Then the following theorem can be derived.

Theorem 14 (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.

The notion of a collection of functions capable of approximating any GNN function is defined as follows.

Definition 15 (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 $\|\mathcal{A} - h_{\mathcal{A},\epsilon}\|_\infty := \sup_{G \in \mathcal{X}} |\mathcal{A}(G) - h_{\mathcal{A},\epsilon}(G)| < \epsilon$

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 proof of Theorem 14 in the Appendix in subsection 1.2.

Sollte ich vielleicht die Definition wie ich Graphen encode und was GNN-Approximating ist auch in die Appendix verschieben um diesen Teil hier übersichtlicher zu machen, oder passt das so und verwirrt nicht zu stark?

By putting all theorems into perspective, 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.

3.1. Proof of Theorem 11: "GNN $\subseteq$ 1-WL+NN"

We will prove Theorem 11 by introducing a couple of small lemmas, which combined prove the theorem. In detail, in Lemma 16, we show the existence of a collection computed by 1-WL+NN that is 1-WL-Discriminating. In Lemmas 17 to 19 we derive properties of 1-WL+NN functions we will use throughout Lemmas 20 to 22 with which we prove the theorem. We took great inspiration for Lemmas 20 to 22 from the proof presented in section 3.1 in the work of Chen et al. [2019].

Der Beweis ist sehr stark von Chen et al. [2019] inspiriert, sodass ich denke, dass er eigenltich komplett korrekt ist. Wäre dennoch super, wenn du einmal detailiert drüber liest und schaust dass mir kein Fehler unterlaufen ist.

Lemma 16. 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 function that returns the number of nodes colored as c. With this, we can construct the collection of functions C as follows:

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

where MLP_{id} is a dummy multilayer perceptron that 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.

Proof of correctness: Let $G_1, G_2 \in \mathcal{X}$ with $G_1 \not\simeq_{1\text{WL}} G_2$. 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 $G_1 \not\simeq_{1\text{WL}} G_2$, 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 17 (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 $G_1 \simeq_{1\text{WL}} G_2$ 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: $\mathcal{B}(\cdot) = \mathsf{MLP} \circ f_{\text{enc}}\{\!\{\text{1-WL}(\cdot)(v) \mid v \in V(\cdot)\}\!\}$. Further, let $G_1, G_2 \in \mathcal{X}$ be arbitrary graphs with $G_1 \simeq_{1\text{WL}} G_2$, then by definition of the relation $\simeq_{1\text{WL}}$ we know that $\text{1-WL}(G_1) = \text{1-WL}(G_2)$. With this, the equivalence follows immediately. □

Lemma 18 (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 $G_1 \simeq_{1\text{WL}} G_2$. Using Lemma 17, we can therefore conclude that: $\mathcal{B}(G_1) = \mathcal{B}(G_2)$. □

Lemma 19 (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 $\mathsf{MLP}^\bullet$ an multilayer perceptron, than the function $\mathcal{B}$ composed of $\mathcal{B}(\cdot) := \mathsf{MLP}^\bullet(h_1(\cdot), \dots, h_n(\cdot))$ is also computable by 1-WL+NN.

Proof Sketch. Assume the above and let $f_1, \dots, f_n$ be the encoding functions, as well as $\mathsf{MLP}_1, \dots, \mathsf{MLP}_n$ be the multilayer perceptrons used by $h_1, \dots, h_n$ respectively. The idea of this proof is that we construct an encoding function f^* that “duplicates” its input and applies each encoding function f_i individually. We also construct a multilayer perceptron MLP^* that takes in the output of f^* and simulates all $\mathsf{MLP}_1, \dots, \mathsf{MLP}_n$ simultaneously. Afterward, the given $\mathsf{MLP}^\bullet$ will be applied on the concatenation of the output of all MLP_i's. See Figure 4 for a sketch of the proof idea. For the complete proof, please refer to the Appendix in subsection 1.1.

$$\begin{tikzcd} G \arrow[r, mapsto, "\text{1-WL}"] & \begin{array}{c} M_G := \\ \{\!\{\text{1-WL}(G)(v) \mid v \in V(G)\}\!\} \end{array} \arrow[r, mapsto, "f^*"] & \begin{bmatrix} f_1(M_G) \\ \vdots \\ f_n(M_G) \end{bmatrix} \arrow[r, mapsto, "\text{MLP}^*"] & \text{MLP}^\bullet\left(\begin{bmatrix} \text{MLP}_1(f_1(M_G)) \\ \vdots \\ \text{MLP}_n(f_n(M_G)) \end{bmatrix}\right) \end{tikzcd}$$

Figure 4.: The proof idea for Lemma 19, how the constructed functions f^* and MLP^* will work on input $G \in \mathcal{X}$. Here we denote with M_G the multiset of colors of the nodes of G after applying the 1-WL algorithm.

Hilft die Grafik? Und fällt auf, dass Sie nicht mit tikz gemacht ist? Schriftart ist ein kleines bisschen größer als der Rest :/

Lemma 20. 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, $G \simeq_{1\text{WL}} G^*$.

Proof. Assume the above. For any $G_1, G_2 \in \mathcal{X}$ with $G_1 \not\simeq_{1\text{WL}} G_2$, 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:

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

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 $G_1 \simeq_{1\text{WL}} G$, then for every function $\overline{h}_{G_1,G_2}$ of the sum with $G_1 \not\simeq_{1\text{WL}} G_2$, we know, using Lemma 17, 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 $G_1 \not\simeq_{1\text{WL}} G$, 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 $\overline{h}_{G_1,G_2}$.

Using Lemma 19, we can conclude that for any $G \in \mathcal{X}$, h_G is computable by 1-WL+NN, as we can encode Equation 0.2 via a multilayer perceptron where the factor "$h_{G_1,G_2}(G_1)$" of Equation 0.1 is just a constant. □

Lemma 21. 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 $G \simeq_{1\text{WL}} G^*$, 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 $G \in \mathcal{X}$: $\varphi_{G^*}(G) = \mathbb{1}_{G \simeq_{1\text{WL}} G^*}$.

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

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

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:

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

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 5 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 $\varphi_{G^*}(\cdot) := \psi_{\delta_{G^*}}(h_{G^*}(\cdot))$. 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 $G \simeq_{1\text{WL}} G^*$, then $h_{G^*}(G) = 0$ resulting in $\varphi_{G^*}(G) = \psi_{\delta_{G^*}}(0) = 1$.
2. If $G \not\simeq_{1\text{WL}} G^*$ 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 19 we can therefore conclude that φ_{G^*} is computable by 1-WL+NN for every graph $G^* \in \mathcal{X}$. □

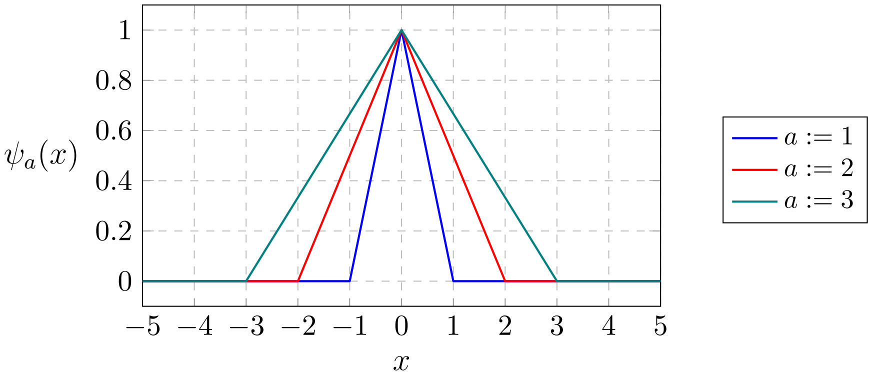


Figure 5.: Illustration of the so-called “bump” function $\psi_a(x)$ used in the proof of Lemma 21 with different exemplary values for a.

Lemma 22. Let $\mathcal{C}$ be a collection of functions from $\mathcal{X}$ to $\mathbb{R}$ computable by 1-WL+NN such 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 GNN is also computable by 1-WL+NN.

Proof. Assume the above. For any function $\mathcal{A}$ computed by a 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:

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

where we denote with $\mathcal{X}/{\simeq_{1\text{WL}}}(G^*)$ the set of all graphs G over $\mathcal{X}$ that are equivalent to G^* according to the $\simeq_{1\text{WL}}$ 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 pair of graphs $G, H \in \mathcal{X}$ with $G \simeq_{1\text{WL}} H$: $\mathcal{A}(G) = \mathcal{A}(H)$. Therefore, we can decompose $\mathcal{A}$ as indicated in Equation 0.4 and encode it using a multilayer perceptron where $\frac{\mathcal{A}(G^*)}{|\mathcal{X}/{\simeq_{1\text{WL}}}(G^*)|}$ is a constant, and $\varphi_{G^*} \in \mathcal{C}$ encodes the indicator function. Combined with the Lemma 19, 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. □

3.2. Proof of Theorem 12: “1-WL+NN $\subseteq$ GNN”

In this section we will prove the converse direction. We start with Lemma 23, where we introduce an upper bound that we will use in Lemma 24 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 25 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 26.

Der Beweis ist komplett von mir selbst, daher wäre es super wenn du hier genauer drauf schauen könntest ob hier alles richtig ist.

Lemma 23. 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, at most n different colors exist 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 24 (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 23, 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 will now construct a GNN with n layers working on input G as follows:

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

Here $\mathsf{RELABEL}_{m,t}$ is a function, parametrized by m and t, that maps the tuples injectively to an integer of the set:

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

This function exists as by the soundness of the upper bound of Lemma 23, the cardinality of its co-domain is greater or equal than the one of its domain. Thereby, and with the injectiveness of $\mathsf{RELABEL}_{m,t}$, we ensure that each GNN layer maps a tuple to a new, previously unused 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 $|V(G)| \leq n$ iterations, we set the number of layers to n, such that we ensure that the coloring computed by this GNN 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.

Ich benutze den Upper-bound um zu argumentieren, dass die $\mathsf{RELABEL}_{m,t}$ so exisitiert. Langt die Existenz oder sollte ich sie auch noch konsturieren?

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 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 25 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 25 (GNN Composition)**.** Let C be a collection of functions computable by GNNs. Further, let $\mathcal{A}_1, \ldots, \mathcal{A}_n \in C$ and $\mathsf{MLP}^\bullet$ a suitable multilayer perceptron, then the function $\hat{\mathcal{A}}(\cdot) := \mathsf{MLP}(\mathcal{A}_1(\cdot), \ldots, \mathcal{A}_n(\cdot))$ is also computable by a GNN.

Proof. Before we begin the proof, we briefly introduce two notations. For any $x \in \mathbb{R}^d$, we will use the notation $x[i]$ to indicate the i.th element of the vector x. Additionally, we 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 Readout function and $f^{(0)}_i$ the input function of $\mathcal{A}_i$.

We will prove the lemma by giving a construction of a GNN model computing $\hat{\mathcal{A}}$. For the ease of readability and to reduce the complexity of the subsequent construction, we assume that for all $\mathcal{A}_i$ its functions $f^{(t)}_{\text{merge},i}, f^{(t)}_{\text{agg},i}$ and $\mathsf{Readout}_i$ map into the one-dimensional space $\mathbb{R}$ for all layers t. With this assumption, we avoid the need for a formal notation of the number of dimensions each of these functions map to.

Let T be the maximum number of layers of all $\mathcal{A}_1, \ldots, \mathcal{A}_n$. We construct the GNN $\hat{\mathcal{A}}$ with T layers, with the input layer working as follows on an input graph G:

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

and each other layer $0 < t \leq T$ utilizing the merge $\hat{f}^{(t)}_{\text{merge}}$ and aggregation $\hat{f}^{(t)}_{\text{agg}}$ functions as constructed in the following:

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

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

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

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

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:

$$\mathsf{Readout}(\{\!\{\hat{f}^{(T)}(v) \mid v \in V(G)\}\!\}) := \mathsf{MLP}^\bullet \circ \begin{bmatrix} \mathsf{Readout}_1(\{\!\{\hat{f}^{(T)}(v)[1] \mid v \in V(G)\}\!\}) \\ \vdots \\ \mathsf{Readout}_n(\{\!\{\hat{f}^{(T)}(v)[n] \mid v \in V(G)\}\!\}) \end{bmatrix}.$$

With this, the proof concludes. Note that this proof can easily be extended to work without the assumption of each function mapping into a one-dimensional space.

Ist die Annahme nachvollziehbar, oder soll ich lieber mit extra indizes den Beweis für den allgemeinen Fall machen?

□

As a consequence of the previous two lemmas, we find ourselves in a similar position as at the beginning of the proof in Section 3.1. Specifically, we have established, through Lemma 24, the existence of a collection C of functions that can be computed by GNNs and can effectively distinguish any pair of graphs that are also distinguishable by the $\mathsf{1\text{-}WL}$ algorithm. Furthermore, with Lemma 25, we have demonstrated that the composition of multiple GNNs and a multilayer perceptron remains computable by a single GNN. Consequently, we can also apply the findings of Lemmas 20 and 21 to GNNs. Thus, we can conclude that for any fixed $G^* \in \mathcal{X}$, the indicator function φ_{G^*} working over $\mathcal{X}$ with:

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

is computable by a GNN.

Ich mache es mir hier sehr einfach, und verwende die Lemmas aus dem Beweis des 1. Theorems. Ist das in Ordnung? Die Lemmas sind ja eig für $\mathsf{1\text{-}WL{+}NN}$ konzipiert, auch wenn man den exact selben Beweise führen würde um das Gleiche für GNNs zu beweisen.

Lemma 26. 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 $\mathsf{1\text{-}WL{+}NN}$ is also computable by a GNN.

Proof. Assume the above. For any function $\mathcal{B}$ computed by $\mathsf{1\text{-}WL{+}NN}$ 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:

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

where we denote with $\mathcal{X}/{\simeq_{1\text{WL}}}(G^*)$ the set of all graphs G over $\mathcal{X}$ that are equivalent to G^* according to the $\simeq_{1\text{WL}}$ relation. Further, with Lemma 17 we know that for any $G_1, G_2 \in \mathcal{X}$ with $G_1 \simeq_{1\text{WL}} G_2 : \mathcal{B}(G_1) = \mathcal{B}(G_2)$.

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 25, we can conclude that $\mathcal{B}$ is computable by a GNN. 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

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

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

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

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

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

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

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

[20] C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel. Faster kernel for graphs with continuous attributes via hashing. In *IEEE International Conference on Data Mining*, pages 1095–1100, 2016.

[21] 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.

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

[23] 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, 2023b.

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

[25] 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.

[26] 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.

[27] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

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

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

[30] 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.

[31] O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[32] 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.

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

[34] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

[35] 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.

A. Appendix Part I

1. Theoretical Connection

1.1. Lemma 19: Composition Lemma for 1-WL+NN

Before we begin with the actual composition proof, we give a formal definition and notation for multilayer perceptrons.

Dieses Lemma ist wie so der Backbone des gesamten Beweises der Hinrichtung. Hoffe die Formaliesierung von den MLP ist in Ordnung. Habe keine Quelle gefunden, die mlps ordentlich definiert, daher ist das selbst ausgedacht.

Definition 27 (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 $\mathsf{MLP} := (\mathsf{MLP})_{i \in [T]}$ where T is the number of layers. For every $i \in [T]$, the i.th layer of the MLP is the i.th item in the finite sequence $(\mathsf{MLP})_i$. Further, all layers are recursively defined on any input v as:

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

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 an input $v \in \mathbb{R}^n$ it is defined as $\mathsf{MLP}(v) := (\mathsf{MLP})_k(v)$.

Having established a formal definition and notation, we will now proof the 1-WL+NN composition lemma.

Der Beweis ist einfach nicht sehr schön, da hier viel mit indexen gehandhabt wird. Daher auch in der Appendix.

Proof of Lemma 19. Let $\mathcal{C}$ be a collection of functions computed by 1-WL+NN, $h_1, \dots, h_n \in \mathcal{C}$, and $\mathsf{MLP}^\bullet$ a multilayer perceptron. Further, 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. As outlined in Figure 4, we will now construct f^* and MLP^*, such that for all graphs $G \in \mathcal{X}$:

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

with which we can conclude that the composition of multiple functions computable by 1-WL+NN, is 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 concat is the concatenation function, concatenating all encoding vectors to one single vector.

Using the decomposition introduced in Definition 27, we can decompose each MLP_i for $i \in [n]$ at layer $j > 0$ as follows: $(\mathsf{MLP}_i)_j(v) := \sigma_{i,j}(W^i_j \cdot (\mathsf{MLP}_i)_{j-1}(v) + b^i_j)$. Using this notation we construct MLP^* as follows:

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

where T is the maximum number of layers of the set of MLP_i's, and $T^\bullet$ is the number of layers of the given $\mathsf{MLP}^\bullet$. Thereby, we define in the first equation the start of the sequence as 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 $\mathsf{MLP}^\bullet$ 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:

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

Unsicher, ob die Aktievierungs -funktion so zusammengebaut werden kann. Beispiel, dann müsste es ja eine Aktievierungsfuntkion σ geben, die auf den ersten 10 Einträge ihres Inputs ReLU anwendet und auf den anderen Einträgen tanh.

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. □

1.2. Theorem 14: Continuous Case: "GNN ⊆ 1-WL+NN"

We will prove Theorem 14 by introducing a definition and then two lemmas using that definition, collectively proving the entire theorem. In particular, we define the property that a collection of functions is able to find any $\simeq_{\mathrm{1WL}}$ equivalence class. In the subsequent Lemma 29, we will show that the quality of 1-WL-Discriminating of a collection implies the ability to locate any $\simeq_{\mathrm{1WL}}$-equivalence class. Finally, with Lemma 30, that this quality further implies the capability of being GNN-Approximating. The overall proof is inspired by the work of [10].

For this proof, we make the major assumption that a continuous 1-WL algorithm exists that operates over continuous values.

Habe sehr lange an dem Beweis gearbeitet und bin sehr stolz drauf. Bin mir sehr sicher dass er richtig ist, aber natürlich nur mit der dieser Annahme klappt. Wäre zu schade, wenn ich den Beweis nicht einbringen kann. Kannst du dennoch vielleicht etwas detailierter auf den Beweis schauen, falls ich doch etwas übersehen habe.

Definition 28 (Locating $\simeq_{\mathrm{1WL}}$-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 $G \simeq_{\mathrm{1WL}} G^* : h_{G^*}(G) = 0$, 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 $G' \in \mathcal{X}/{\simeq_{\mathrm{1WL}}}(G)$ in the equivalence class of G such that $\|G' - G^*\|_2 < \epsilon$

we say $\mathcal{C}$ is able to locate every $\simeq_{\mathrm{1WL}}$ 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 ($G \simeq_{\mathrm{1WL}} G^*$), only a small loss if G is close to a graph in the $\simeq_{\mathrm{1WL}}$ equivalence class of G^* (the loss is upper bounded by δ_{G^*}), and an arbitrary loss otherwise.

Lemma 29. 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 $\simeq_{\mathrm{1WL}}$ 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 $G \not\simeq_{\mathrm{1WL}} G^*$, 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 $G \not\simeq_{\mathrm{1WL}} G^*$ a set A_{G,G^*} as follows:

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

where $(a \pm b)$ is the open set $(a-b, a+b)$ over $\mathbb{R}$. One can use Figure A.1 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 $G^* \notin A_{G,G^*}$.

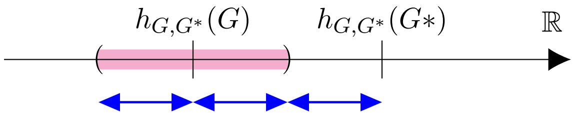


Figure A.1.: An illustration to better understand the proof. The set A_{G,G^*} consists of all graphs G that are mapped by h_{G,G^*} into the pink area, which size depends on the distance between the image of G and G^* under h_{G,G^*}. Moreover, the blue distances all have the same length.

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 norm.

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 17 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:

   $$\delta_{G^*} \coloneqq \frac{1}{2} \min_{\substack{G_0 \in \mathcal{X} \\ G_0 \not\simeq_{1\text{WL}} G^*}} |h_{G_0,G^*}(G_0) - h_{G_0,G^*}(G^*)|.$$

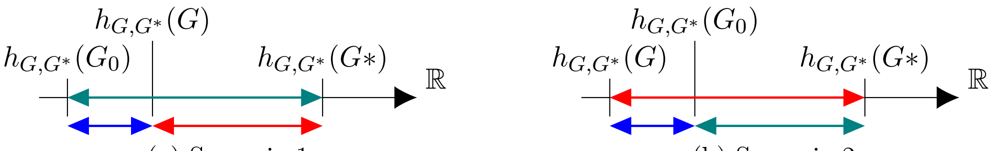
   Prove by contraposition: Assume that for every graph $G' \in \mathcal{X}/\simeq_{1\text{WL}}(G)$: $\|G' - G^*\|_2 \geq \epsilon$. Hence, $G \notin \bigcup_{G' \in \mathcal{X}/\simeq_{1\text{WL}}(G^*)} A_{G',G^*}$ (not in the union of l_2 balls of size ϵ around all graphs of the equivalence class of G^*). 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:

$$\begin{aligned}
\overline{h}_{G_0,G^*}(G) &= |h_{G_0,G^*}(G) - h_{G_0,G^*}(G^*)| \\
&\geq |h_{G_0,G^*}(G_0) - h_{G_0,G^*}(G^*)| - |h_{G_0,G^*}(G) - h_{G_0,G^*}(G_0)| \qquad \text{(A.5)} \\
&\geq \frac{1}{2}|h_{G_0,G^*}(G_0) - h_{G_0,G^*}(G^*)| \qquad \text{(A.6)} \\
&\geq \frac{1}{2} \min_{\substack{G_0 \in \mathcal{X} \\ G_0 \not\simeq_{1\mathrm{WL}} G^*}} |h_{G_0,G^*}(G_0) - h_{G_0,G^*}(G^*)| =: \delta_{G^*}
\end{aligned}$$

To understand these inequalities, it helps to visualize them, hence see Figure A.2. We will try to give an explanation now, using the colored distances depicted in Figure A.2. 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.2.: 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 29 and Figure A.2.

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

As a final note, we can construct a multilayer perceptron $\mathsf{MLP}_{h_{G^*}}$ computing the function $h_{G^*}(\cdot)$ as in Equation (A.1). The $\mathsf{MLP}_{h_{G^*}}$ gets as input a vector of the output of the finite set of functions $\{\overline{h}_{G_0,G^*}\}_{G_0 \in \mathcal{X}_0,\ G_0 \not\simeq_{1\mathrm{WL}} G^*}$ 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 19, we can conclude that $h_{G^*}(\cdot)$ is computable by 1-WL+NN.

$\square$

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 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 GNN-Approximating.

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, which we will fix for the remainder of the proof as well. 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.3 for a better understanding.

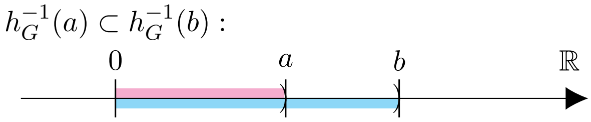


Figure A.3.: Illustration of the set $h_G^{-1}(\cdot)$ for two arbitrary values a, b with $a < b$. The pink area visualizes the open interval $[0, a)$, and similarly in blue for $[0, b)$.

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

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

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:

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

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 $\hat{A}$ is computable by $\mathsf{1\text{-}WL{+}NN}$. For this we construct a multilayer perceptron with three layers and then conclude with Lemma 19 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 $\mathsf{MLP}_{\hat{\mathcal{A}}}$ 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. □