An In-Depth Analytical Framework for Investigating Representations learned by Graph Neural Networks

From the faculty of Mathematics, Physics, and Computer Science 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 Definition

We will quickly go over some fundamental definitions. More!!!

1.1 Basics

Definition 1 (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 $\text{MLP} := (\text{MLP})_{i\in[k]}$ 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:

$$\begin{aligned}(\text{MLP})_1(v) &:= v\\ (\text{MLP})_{i+1}(v) &:= \sigma(W_i \cdot (\text{MLP})_i(v) + b_i), \quad \forall i \in [k-1]\end{aligned}$$

where σ 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 $\text{MLP}(v) := (\text{MLP})_k(v)$.

Permutation-invariance and -equivariance

We use S_n to denote the symmetric group over the elements $[n]$ for any $n > 0$. 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 $V(G_\pi) = \{\pi(1), \dots, \pi(n)\}$ 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 2 (Permutation Invariant)**.** Let $f : \mathcal{G} \to \mathcal{X}$ be an arbitrary function and let $V(G) = [n]$ for some $n \in \mathbb{N}$. The function 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 3 (Permuation Equivariant)**.** Let $f : \mathcal{G} \to \mathcal{X}$ be an arbitrary function and let $V(G) = [n]$ for some $n \in \mathbb{N}$. The function f is *permuation-equivariant* if and only if for all $G \in \mathcal{G}$ where $n_G :=\mid V(G) \mid$ and for every $\pi \in S_{n_G}$: $f(G) = \pi^{-1} \cdot f(\pi \cdot G)$.

1.2 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

Definition 4 (1-WL Algorithm)**.** Let $G = (V, E, l)$ be a graph, then in each iteration i, the 1-WL computes a node coloring $C_i : V(G) \to \mathbb{N}$, which depends on the coloring of the neighbors and the node itself. In iteration $i = 0$, the initial coloring is $C_0 = l$ or if l is non-existing $C_0(v) = c$ for all $v \in V(G)$ 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. Although this is not a formal restriction by the inventors, we further require the function to always map to the next minimal natural number. Thereby we can contain the size of the codomain of each coloring for all iterations. 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 algorithm always terminates after $n_G := |V(G)|$ iterations (**?**).

For an illustration of this coloring algorithm, see Figure 2. Moreover, based on the work of **?** about efficient refinement strategies, **?** proved that the stable coloring C_∞ can be computed in time $\mathcal{O}(|V(G)| + |E(G)| \cdot \log |V(G)|)$.

Bring many infos outside of the definiton. And add that we call this algorihm 1-WL.

The Weisfeiler-Leman Graph Isomorphism Test

Definition 5 (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:

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

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 1 which was discovered and proven by **?**.

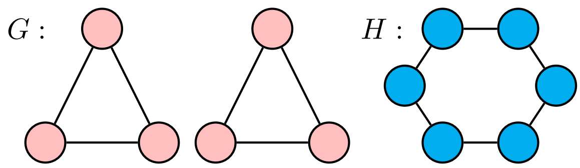

Figure 1: An example of two graphs G and H that are non-isomorphic but cannot be distinguished by the 1-WL isomorphism test.

1.3 1-WL+NN

Text ABC

Here to do

Definition 6 (1-WL+NN)**.** We say the function $\mathcal{B} : \mathcal{G} \to \mathbb{R}^m$ is computable by 1-WL+NN, if it can be compromised as $\mathcal{B}(\cdot) = \text{MLP} \circ f_{\text{enc}} \circ \text{1-WL}(\cdot)$, where f_{enc} is an encoding function that maps graph coloring to fixed-sized vectors, and MLP is a multilayer perceptron.

f must be permutation invariant?

As a concrete example of a collection of functions computable by 1-WL+NN we will introduce the collection $\mathfrak{B}_k$ that is parametrized by $k \in \mathbb{N}_{\geq 1}$. All functions $\mathcal{B} \in \mathfrak{B}_k$ use the *counting-encoding* function f_{count} as their encoding function, and are constrained in their domain to only work over a subset $\mathcal{X}$ of $\mathcal{G}$. We will define this particular encoding function in the following:

Definition 7 (Counting Encoding Functions)**.** For $k \in \mathbb{N}_{\geq 1}$, let

$$\mathcal{X} = \{G \in \mathcal{G} \mid \forall x \in V(G) \cup E(G) : l_G(x) \leq k\} \subset \mathcal{G}$$

be the set of all graphs, where the label alphabet Σ of the respective label function l is bounded with $\Sigma \subseteq [k]$. We define the *counting-encoding* function $f_{\text{count}} : \text{1-WL}(\mathcal{X}) \to \mathbb{N}^K$ as the function that maps a graph coloring C_∞ of a graph $G \in \mathcal{X}$ to a vector $v \in \mathbb{N}^K$ such that the c.th component of v is equal to the number of occurrences of the color c in the coloring C_∞. More formally, for $G \in \mathcal{X}$ let C_∞ be the final coloring upon the termination of the 1-WL algorithm on G and h_{G,C_∞} the respective color histogram. Then f_{count} maps C_∞ to a vector $v \in \mathbb{N}^K$, such that for all $c \in [K] : v_c = h_{G,C_\infty}(c)$, where v_c denotes the c.th component of the vector v. Important to note, due to the bounded label alphabet Σ of all graphs $G \in \mathcal{X}$ by the parameter k, there exists a minimal K for the codomain $\mathbb{N}^K$ of f_{count}, such that f_{count} is well-defined on all graphs $G \in \mathcal{X}$.

To illustrate how this encoding function works and why we coined it *counting-encoding*, we will quickly introduce an example graph G. In Figure 2, we give a visual representation of G and its stable coloring after applying the 1-WL algorithm to it. The *counting-encoding* function f_{count} 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 $v \in \mathbb{N}^K$ with $v = (0, 2, 1, 1, 0, \ldots, 0)^T$, since color 2 appears two times, while color 3 and 4 occur only once. All other components of v are set to 0.

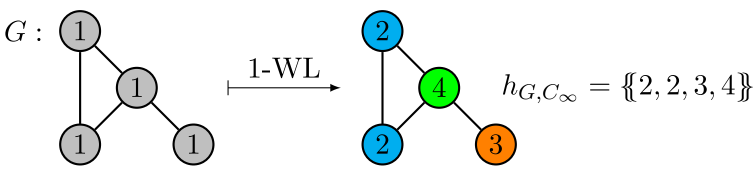

Figure 2: An example of the final coloring computed by applying the 1-WL algorithm on the graph G. The graph G consists of 4 nodes with all their labels being initially set to 1. Note that each label corresponds to a color, which we have also plotted for illustration purposes.

1.4 Graph Neural Networks (Message Passing)

Let $G = (V, E, l)$ be an arbitrary graph. A Graph Neural Network (GNN) is a composition of multiple layers where each layer t passes a vector representation of each node v or edge e through $f^{(t)}(v)$ or $f^{(t)}(e)$ respectively and retrieves thereby a new graph that is structurally identical but has new feature information. 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.

Better intro text

Definition 8 (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, 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 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:

$$\mathcal{A} : \mathcal{G} \to \mathcal{Y} : x \mapsto \mathsf{READOUT} \circ f^{(K)} \circ \ldots \circ f^{(0)}(x),$$

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

$$\mathcal{A} : \mathcal{G} \to \mathcal{Y} : x \mapsto f^{(K)} \circ \ldots \circ f^{(0)}(x).$$

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 **?** and **?**.

To demonstrate what kind of functions are typically used, we provide functions used by **?** for a node classification:

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

where σ is a non-linear element wise activation function; W_{merge}, W_{pool} are trainable matrices, b a trainable vector and concat the concatenation function.

1.5 Important for later

Definition 9 (1-WL Relation)**.** For any graphs G, H 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 $G \not\simeq H$.

Definition 10 (1-WL-Discriminating)**.** Let $\mathcal{C}$ be a collection of permutation invariant functions from $\mathcal{X}^{n\times n}$ 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 $f(G_1) \neq f(G_2)$.

Definition 11 (GNN-Approximating)**.** Let $\mathcal{C}$ be a collection of permutation invariant functions from $\mathcal{X}^{n\times n}$ 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 $\|\mathcal{A} - h_{\mathcal{A},\epsilon}\|_\infty := \sup_{G\in\mathcal{X}} |f(G) - h_{\mathcal{A},\epsilon}(G)| < \epsilon$

2 Theorems

Throughout this thesis we will concentrate on a finite collection of finite graphs which we will denote with $\mathcal{X} \subset \mathcal{G}$.

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 GNNs, then $\mathcal{C}$ is also computable by 1-WL+NN.

Theorem 13 (Finite Case: "GNN $\subseteq$ 1-WL+NN")**.** 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.

2.1 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 14 we show the existence of a collection computed by 1-WL+NN that is 1-WL-Discriminating. In Lemmas 15 to 17 we derive properties of 1-WL+NN functions we will use throughout Lemmas 18 to 20 with which we prove the theorem. We took great inspiration for Lemmas 18 to 20 from the proof presented in section 3.1 in the work of **?**.

Lemma 14. 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 consider the collection $\mathfrak{B}_k$ (Definition 7) of functions from $\mathcal{X}$ to $\mathbb{R}$ computed by 1-WL+NN, where we choose k as follows:

$$k := \max(\{l_G(v) \mid G \in \mathcal{X}, v \in V(G)\}),$$

the largest label of any node of any graph in $\mathcal{X}$. Note that we can compute k, since $\mathcal{X}$ is finite.

Let $G_1, G_2 \in \mathcal{X}$ such that the 1-WL isomorphism test concludes non-isomorphic ($G_1 \not\simeq_{1\text{WL}} G_2$). Let C_1, C_2 be the final coloring 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 $h_{G_1,C_1}(c) \neq h_{G_2,C_2}(c)$. If we now consider as multilayer perceptron $\text{MLP}_c : \mathbb{N}^K \to \mathbb{R}, v \mapsto W \cdot v$ with $W \in \mathbb{N}^{1\times K}$ such that $W_{1,c} := 1$ and $W_{1,i} := 0$ for all $i \in [K] \setminus \{c\}$, we can construct $\mathcal{B}$ as $\mathcal{B}(\cdot) := \text{MLP} \circ f_{\text{count}} \circ \text{1-WL}(\cdot)$, such that $\mathcal{B} \in \mathfrak{B}_k$. Then $\mathcal{B}(G) := h_{G,(C_\infty)_G}(c)$, such that we can conclude $\mathcal{B}(G_1) \neq \mathcal{B}(G_2)$,and since G_1, G_2 were arbitrary chosen, we can conclude the proof. $\square$

Big K explaining, linking to defintion of $\mathfrak{B}$

Lemma 15 (1-WL+NN Equivalence)**.** Let $\mathcal{C}$ be a collection of functions computable by 1-WL+NN, then for every function $\mathcal{B} \in \mathcal{C}$ and 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 $\mathcal{C}$ be a collection of functions computed by 1-WL+NN. Let $\mathcal{B}$ be an arbitrary function in $\mathcal{C}$, then $\mathcal{B}$ is comprised as follows: $\mathcal{B}(\cdot) = \text{MLP} \circ f_{\text{enc}} \circ \text{1-WL}(\cdot)$. 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 immediatly. $\square$

Lemma 16 (1-WL+NN Permuation Invariance)**.** Let $\mathcal{C}$ be a collection of functions computable by 1-WL+NN, then every function $\mathcal{B} \in \mathcal{C}$ 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 15, we can therefore conclude that: $\mathcal{B}(G_1) = \mathcal{B}(G_2)$. $\square$

Lemma 17 (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 $\text{MLP}^\bullet$ an multilayer perceptron, than the function $\mathcal{B}$ composed of $\mathcal{B}(\cdot) := \text{MLP}(h_1(\cdot), \dots, h_n(\cdot))$ 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 $\text{MLP}_1, \dots, \text{MLP}_n$ 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 maps a coloring C_∞ to a concatenation of the vectors obtained when applying each encoding function f_i individually. Additionally, we construct a multilayer perceptron MLP^* that takes in this concatenation of vectors and simulates all $\text{MLP}_1, \dots, \text{MLP}_n$ simultaneously on their respective section of the encoding vector of f^*, and applies afterwards the given $\text{MLP}^\bullet$ on the concatenation of the output of all MLP_i's. See Figure 3 for a sketch of the proof idea. A complete proof can be found in the Appendix, as this proof is very technical and not that interesting.

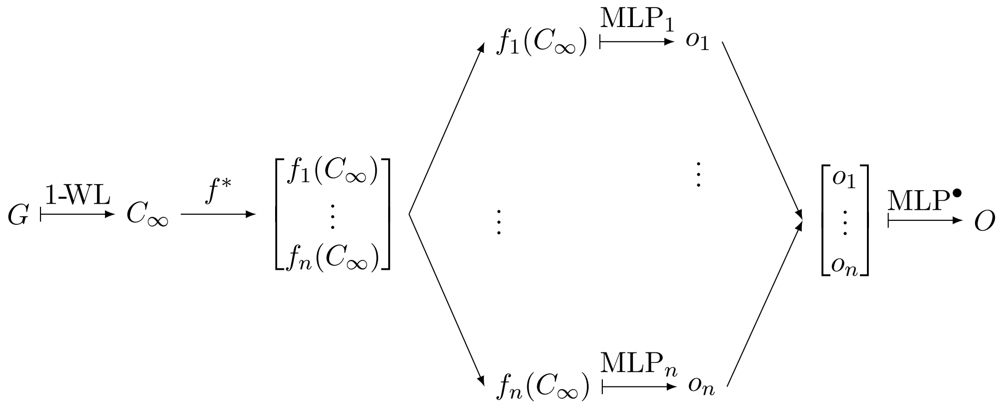

Figure 3: Proof idea for Lemma 17, how the constructed functions f^* and MLP^* will work on input G. Here we denote with C_∞ the final coloring computed by the 1-WL algorithm when applied on G. Further, we let o_i be the output computed by MLP_i on input $f_i(C_\infty)$.

□

Lemma 18. 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. For any $G_1, G_2 \in \mathcal{X}$ with $G_1 \not\simeq_{\text{1WL}} G_2$ let $f_{G_1,G_2} \in \mathcal{C}$ be the function distinguishing them, with $f_{G_1,G_2}(G_1) \neq f_{G_1,G_2}(G_2)$. We define the function $\overline{f}_{G_1,G_2}$ working over $\mathcal{X}$ as follows:

$$\begin{aligned} \overline{f}_{G_1,G_2}(\cdot) &= |f_{G_1,G_2}(\cdot) - f_{G_1,G_2}(G_1)| \\ &= \max(f_{G_1,G_2}(\cdot) - f_{G_1,G_2}(G_1)) + \max(f_{G_1,G_2}(G_1) - f_{G_1,G_2}(\cdot)) \end{aligned} \tag{0.1}$$

Note, that in the formula above "$f_{G_1,G_2}(G_1)$" is a fixed constant and the resulting function $\overline{f}_{G_1,G_2}$ 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:

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

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 $\overline{f}_{G_1,G_2}$ of the sum with $G_1 \not\simeq_{1\text{WL}} G_2$, we know, using Lemma 15, that $\overline{f}_{G_1,G_2}(G^*)$ is equal to $\overline{f}_{G_1,G_2}(G_1)$ which is by definition 0, such that $h_{G_1}(G^*) = 0$.
2. If $G_1 \not\simeq_{1\text{WL}} G^*$, then $\overline{f}_{G_1,G^*}(G^*)$ is a summand of the overall sum, and since $\overline{f}_{G_1,G^*}(G^*) > 0$, we can conclude $h_{G_1}(G*) > 0$ due to the non-negativity of each function $\overline{f}$.

This function can be encoded in an MLP by replacing the max terms of the last line in Equation 0.1 by the activation function ReLU. Therefore, we can conclude with Lemma 17 that for every graph G, h_G is also 1-WL+NN computable. □

Lemma 19. 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 will be similar to the indicator function. We define this function for parameter $a \in \mathbb{R}$ with $a > 0$ as:

$$\psi_a(x) := \max(\frac{x}{a} - 1,\ 0) + \max(\frac{x}{a} + 1,\ 0) - 2 \cdot \max(\frac{x}{a},\ 0).$$

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 4 in the Appendix 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 $|h_G(G^*)| > \delta_G$ resulting in $\varphi_G(G^*) = 0$.

Note that we can encode φ_G via a single MLP layer, where δ_G is a constant and the max operator is replaced by the non-linear activation function ReLU. With Lemma 17 we can therefore conclude that φ_G is computable by 1-WL+NN for every graph $G \in \mathcal{X}$. □

Lemma 20. 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 $\forall G^* \in \mathcal{X} : \varphi_G(G^*) = \mathbb{1}_{G \simeq_{1\text{WL}} G^*}$, then every permutation invariant function computable by a GNN is also computable by 1-WL+NN.

Proof. Assume the above. For any permutation invariant 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:

$$\begin{aligned}
\mathcal{A}(G^*) &= \Big(\frac{1}{|\mathcal{X}/{\simeq_{1\mathrm{WL}}}(G^*)|} \sum_{G \in \mathcal{X}} \mathbb{1}_{G \simeq_{1\mathrm{WL}} G^*} \Big) \cdot \mathcal{A}(G^*) \\
&= \frac{1}{|\mathcal{X}/{\simeq_{1\mathrm{WL}}}(G^*)|} \sum_{G \in \mathcal{X}} \mathcal{A}(G) \cdot \mathbb{1}_{G \simeq_{1\mathrm{WL}} G^*} \\
&= \sum_{G \in \mathcal{X}} \frac{\mathcal{A}(G)}{|\mathcal{X}/{\simeq_{1\mathrm{WL}}}(G)|} \cdot \varphi_G(G^*)
\end{aligned} \tag{0.2}$$

with $\mathcal{X}/{\simeq_{1\mathrm{WL}}}(G^*)$ we denote the set of all graphs G over $\mathcal{X}$ that are equivalent to G^* according to the $\simeq_{1\mathrm{WL}}$ 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 graph $G, H \in \mathcal{X}$ with $G \simeq_{1\mathrm{WL}} H$: $\mathcal{A}(G) = \mathcal{A}(H)$. Therefore, we can decompose $\mathcal{A}$ as stated in Equation 0.2 in a single MLP layer with $\frac{\mathcal{A}(G)}{|\mathcal{X}/{\simeq_{1\mathrm{WL}}}(G)|}$ being constants and $\varphi_G \in \mathcal{C}$ encoding the indicator function. Combined with the Lemma 17, 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 $\mathcal{X}/{\simeq_{1\mathrm{WL}}}(G)$ well-defined for all graphs. □

2.2 Proof of Theorem 13

Lemma 21. Let G be an arbitrary graph with $n := |V(G)|$ the number of nodes and C a 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 we can sum over each cardinality of a multiset over n different colors. In the end, we simply combine both results by multiplying both together. □

Lemma 22. Let $\mathcal{X}$ be a finite collection of graphs. Then there exists a function $\mathcal{A}$ computable by a GNN, that computes a coloring C on input G, that is equivalent under automorphism to the final coloring C' computed by the 1-WL algorithm when applied on G. Formally, there exists an bijective function $\varphi : \mathbb{N} \to \mathbb{N}$, such that for every $c \in \mathbb{N} : |h_{G,C}(c)| = |h_{G,C'}(\varphi(c))|$

Proof. Let $n := \max\{|V(G)| \mid G \in \mathcal{X}\}$ be the maximum number of nodes of a graph in $\mathcal{X}$. Using Lemma 21, 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 the GNN with n layers, where each layer $t > 0$ works as follows for any $v \in V(G)$:

$$f^{(t)}(v) = h_{m,t}(f^{(t-1)}(v),\ \{\!\{f^{(t-1)}(u) \mid u \in \mathcal{N}(v)\}\!\}).$$

Here $h_{m,t}$ is a function that maps the tuples injectively to an integer of the set $\{(t+1) \cdot m + 1, \ldots, (t+2) \cdot m\}$. 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 sinlge 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. $\square$

Frage im Allgemeinem Soll ich hier lieber einen konstruktive Beweis schreiben, oder langt so ein existenziellen Beweis?

Appendix

Figures and graphs

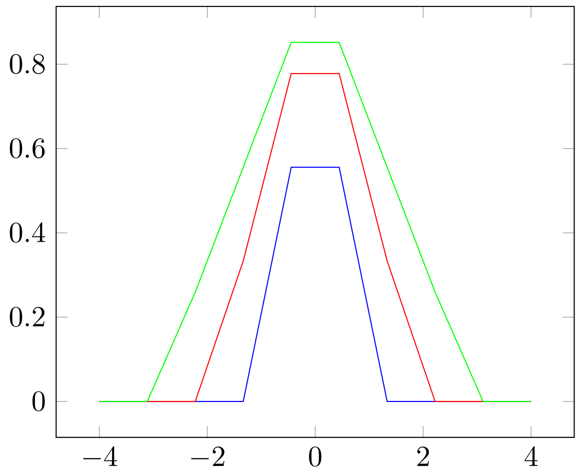

Figure 4: Illustration of the so-called "bump" function $\psi_a(x)$ used in the proof of Lemma 19. Here the colors of the displayed functions correspond to the parameter a set to $a := 1$ in blue, $a := 2$ in red and $a := 3$ in green.

Proofs

Lemma 17. Let $\mathcal{C}$ be a collection of functions computed by 1-WL+NN, $h_1, \ldots, h_n \in \mathcal{C}$, and $\text{MLP}^\bullet$ 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 $G \in \mathcal{X}$:

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

such that 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 input C_∞:

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

where $\mathsf{conccat}$ is the concatenation functions, concatenating all encoding vectors to one single vector.

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

MLP* as follows:

$$
\begin{aligned}
&(\text{MLP}^*)_1(v) := v \\
&(\text{MLP}^*)_{j+1}(v) := \sigma(W_j^* \cdot (\text{MLP}^*)_j(v) + \mathsf{concact}(\begin{bmatrix} b_j^1 \\ \vdots \\ b_j^n \end{bmatrix})) && ,\forall j \in [k] \\
&(\text{MLP}^*)_{j+k+1}(v) := (\text{MLP}^\bullet)_{j+1}(v) && ,\forall j \in [k^\bullet - 1]
\end{aligned}
$$

where k is the maximum number of layers of the set of MLP$_i$'s, $k^\bullet$ is the number of layers of the given MLP$^\bullet$ and σ an element wise activation function. 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 MLP$^\bullet$ to the end. Further, we define the weight matrix W_j^* as follows:

$$
W_j^* := \begin{bmatrix} W_j^1 & 0 & \dots & 0 \\ 0 & W_j^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & W_j^n \end{bmatrix},
$$

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_j^i not exist, because it has less than j layers, we use for W_j^i the identity matrix I_m where m is the dimension of the output computed by MLP$_i$.

Then $\mathcal{B}(\cdot) :=$ MLP$^* \circ f^* \circ$1-WL$(\cdot)$ is 1-WL+NN computable and equavalent to MLP$^\bullet(h_1, \dots, h_n)$ $\square$

fine tune.