Introduction to GNNs with a Focus on Graph Isomorphism Testing

Nelly Körling

June 2025

Introduction

Data sets depicting information such as social networks and molecules are best represented by graphs where data points (nodes) are connected according to their relationships with each other (edges). Graphs are non-Euclidean, meaning they don't fit into the regular grids or sequences that traditional neural networks (like CNNs or RNNs) are designed to handle.

Graph neural networks (GNNs) are a class of models developed to work with graph-structured data. GNNs take advantage of properties specific to graphs to complete tasks on a node-, edge-, or graph-level. Tasks can include classifying users (nodes) in a social network, predicting types of bonds (edges) between atoms in a molecule, and determining the properties of a molecule (graph).

When working with finding embeddings for nodes, edges, and graphs, we would like to know that our model possesses satisfactory capacity to distinguish different graph structures. In other words, structurally identical graph structures should get assigned the same embeddings. Central to this problem is the Weisfeiler–Lehman (WL) test, a classical algorithm for testing graph isomorphism (whether two graphs are structurally the same, even if their node labels differ). It turns out that certain GNN models can be proved to be as powerful as the WL test in distinguishing graph structures.

In what follows, a mathematical background will first be introduced to motivate the most basic GNN model. This model serves as a starting point for examining how the WL-test can be used to analyze and compare the expressive power of different GNN architectures. The relationship between GNNs and the WL-test will be detailed with reference to some central articles written on the topic.

Relevant Mathematical Background

Graphs

A *graph* $G = (V, E)$ is defined by its set of nodes/vertices V and its set of edges $E \subseteq V \times V$ that connect pairs of nodes. If there is an edge between *vertices* $u \in V$ and $v \in V$, this *edge* is denoted by (u, v). In the context of machine learning on graphs, we will be mostly concerned with *simple graphs*; graphs with at most one edge between each pair of nodes, no edge between a node and itself, and where all edges are *undirected* ($(u, v) \in E \Leftrightarrow (v, u) \in E$).

An *adjacency matrix* A is a common representation of a graph that aims to describe which nodes in the graph are connected by an edge. If there are $|V| = n$ nodes in the graph, the size of this matrix is $n \times n$, with a row and column dedicated to each vertex. The value at $A[u, v]$, for vertices $u \in V$ and $v \in V$, is 1 if there exists an edge between u and v, and 0 otherwise. The adjacency matrix is symmetric if the graph is undirected.

In many applications, nodes and/or edges will be associated with feature vectors. For instance, the node feature vectors could be d-dimensional real-valued vectors $x_v \in \mathbb{R}^d$ for every $v \in V$. These are usually aggregated into a node feature matrix, represented by $X \in \mathbb{R}^{n \times d}$. Similar representations are used for edge feature vectors $e_{uv} \in \mathbb{R}^{d'}$.

The concept of *graph isomorphism* will be important in exploring the structure and limitations of GNNs. Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there exists a bijection $\phi : V \to V'$ such that:

$$(u, v) \in E \Leftrightarrow (\phi(u), \phi(v)) \in E'. \tag{1}$$

Informally, if two graphs are isomorphic, they are structurally identical, but with differently labeled nodes.

Weisfeiler-Lehman Isomorphism Testing

The *Weisfeiler-Lehman (WL) algorithm* is a heuristic to test for graph isomorphism between two given input graphs G_1 and G_2. The 1-WL test is the one-dimensional version of the WL algorithm and consists of the following steps:

1. Assign initial labels $l^{(0)}_{G_i}(v)$ to all nodes v in both graphs G_1 and G_2. The label of a node could be its degree or its features.

2. Iteratively assign new labels $l^{(i)}_{G_i}(v)$ to each node in each graph by hashing the node's current label $l^{(i-1)}_{G_i}(v)$ with the multi-set of its neighbors' current labels, $\{\{l^{(i-1)}_{G_i}(u)\ \forall u \in N(v)\}\}$.

3. Repeat step 2 until the labels converge. Convergence is usually defined as when the labels of all vertices do not change between iterations of step 2.

4. Summarize the node labels in each graph by the following multi-sets:

$$L_{G_j} = \{\{l^{(i)}_{G_j}(v), \forall u \in V_j, i = 0, ..., K-1\}\} \tag{2}$$

For $j = 1, 2$.

If these multi-sets differ, the graphs are certainly non-isomorphic. However, equivalency between multi-set representations for two graphs does not guarantee isomorphism. There exist non-isomorphic graphs which the 1-WL cannot distinguish.

In the 1-WL test, we consider labels of individual nodes. As an extension, the k -WL algorithms consider labels of subgraphs of size k. The $(k+1)$-WL algorithm is strictly more powerful at distinguishing isomorphic graphs than the k-WL algorithm for $k \geq 1$. In other words, as k increases, an increasing number of pairs of non-isomorphic graphs can be distinguished from each other.

Permutation Invariance

Since node indices are arbitrary, any learning on graphs must be permutation-aware. We can classify a function as either *permutation invariant* or *permutation equivariant*:

1. A function f is *permutation invariant* if:

$$f(\pi \cdot G) = f(G) \tag{3}$$

for any permutation π of node labels. Permutation invariance is required for graph-level learning tasks because different node orderings should not produce different outcomes for the same graph.

2. A function f is *permutation equivariant* if:

$$f(\pi \cdot G) = \pi \cdot f(G) \tag{4}$$

for any permutation π of node labels. Permutation equivariance is required for node-level learning tasks, since a change in node ordering should be reflected in the output, which is node-specific.

Multilayer Perceptrons and the Universal Approximation Theorem

Multilayer Perceptrons (MLPs) can be said to be the most fundamental architecture among neural networks. They play a crucial role in understanding the representational power of neural networks and serve as a conceptual building block for more advanced models such as convolutional and graph neural networks.

An MLP consists of a sequence of fully connected layers, each applying a linear transformation followed by a non-linear activation function. Formally, given an input vector $x \in \mathbb{R}^d$, the l-th layer of an MLP computes:

$$h^{(l)} = \sigma(W^{(l)} h^{(l-1)} + b^{(l)}), \tag{5}$$

where σ is a non-linear activation function such as ReLU or sigmoid. If there are a total of L layers in the MLP, the output (prediction) will be

$$\hat{y} = W^{(L+1)} h^{(L)} + b^{(L+1)} \tag{6}$$

In modern deep learning, MLPs are rarely used on their own, but they can be important components of various neural network models due to the Universal Approximation Theorem. This theorem guarantees the existence of an MLP that can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to some arbitrary accuracy. MLPs are sometimes incorporated into more advanced neural network models because they can be used to learn complex functions within, for example, a embedding update function.

Graph Neural Networks

Traditional neural networks such as CNNs and RNNs that are used for learning on grids or text, do not perform well when given a graph as an input. They have difficulty synthesizing information from nodes that are connected to an varying number of neighbors in no fixed order, and may not know how to successfully aggregate node and edge features. These difficulties in applying traditional neural networks to graphs inspired the development of the graph neural network (GNN). This type of neural network is designed to solve learning tasks where graphs are given as input by taking advantage of properties unique to graphs. GNNs can be used for node, edge, or graph classification. Node classification tasks entails a GNN predicting labels for nodes. An example might be classifying users in a social network. Edge classification may be an binary problem, such has predicting whether or not a connection exists between two nodes. Graph classification, much like node classification, is assigning a label to an entire graph. For example, classifying molecules into groups, such as toxic or non-toxic. In what follows, the focus will lie in graph and node classification.

Basic GNNs

The simplest way to build a GNN is by using message passing, a way of aggregating information from a node's neighbors to update the node's embedding. The general framework is conveyed by the following update expression for a node v's embedding at layer t:

$$h_v^{(t)} = \text{UPDATE}^{(t)}(h_v^{(t-1)}, \text{AGGREGATE}^{(t)}(\{h_u^{(t-1)}, \forall u \in N(v)\}), \qquad (7)$$

where $h_v^{(t)}$ is the embedding of node v as iteration t, and $N(v)$ denotes the set of neighbors of v. It is important that the AGGREGATE function is permutation-invariant to ensure that the representation update for each node does not depend on the ordering of its neighbors. Common AGGREGATE functions include max, mean, and sum. The message passing is repeated for T layers. At each layer, information is propagated one step further across the graph. After T layers, each node's representation incorporates information from all nodes in its T-hop neighborhoods.

We can reformulate the message passing expression to liken a standard neural network layer, explicitly including learnable weights and biases:

$$h_v^{(t)} = \sigma(W_1^{(t)} h_v^{(t-1)} + W_2^{(t)} \sum_{u \in N(v)} h_u^{(t-1)} + b^{(t)}). \qquad (8)$$

Sometimes, the bias $b^{(t)}$ is excluded for simplicity. Weights (and the bias) can be shared across message passing layers or trained separately for each layer.

While GNNs designed for node classification use variations of the above update expressions, graph classification requires some pooling of node embeddings to get an embedding for the entire graph. Simple pooling methods include taking the mean or sum of all node embeddings in the graph. Importantly, these functions are permutation-invariant, so graphs that have the same structure but a different ordering of nodes will get the same embedding. If the message passing is run for T iterations, the graph embedding would be:

$$h_G = \sum_{v \in V} h_v^{(T)}. \tag{9}$$

Training GNNs

To know how to train a GNN, it is important to observe how loss functions are defined and how training data is sampled. There are, of course, other important aspects of training GNNs. These will be omitted here since the focus of this paper is the theoretical connection between the WL isomorphism test and GNNs. For node classification, training is usually fully supervised manner and the loss function is:

$$\mathcal{L} = -\sum_{u \in V_{train}} \log(\text{softmax}(z_u, y_u), \tag{10}$$

where z_u is the final output of the GNN and y_u is the embedding of node u in the training data. The common loss function used for graph classification is much the same, but with embeddings z_G and y_G for training data $\{G_1, G_2, ..., G_n\}$.

Constructing useful training data sets for GNNs, especially those concerned with node classification in a single graph, can be tricky. Choosing a random set of nodes in the vertex set of a graph can lead to disconnected components, limiting the propagation of information between nodes through message passing. This would undermine the structure of the original graph. Therefore, learning on a random subset of nodes would clearly be inefficient. One way of circumventing this problem (solution proposed by Hamilton 2017) is to select a target node in the graph and then recursively sample the neighbors of these nodes. This would ensure the creation of a connected component that could be used for training.

Connection to Weisfeiler-Lehman Isomorphism Testing

Message Passing GNNs and Weisfeiler-Lehman Isomorphism Testing

When designing GNN models, a natural goal is ensure that a model produces similar embeddings for similar graphs. Importantly, they should different embeddings for non-isomorphic graphs. As previously discussed, a common tool for studying graph isomorphism is the Weisfeiler–Lehman test, which iteratively refines node labels based on their neighborhoods. Since both the WL test and message passing GNNs rely on aggregating information from neighboring nodes, it makes sense to compare their behaviors. In fact, we can measure the expressive power of a GNN by asking whether it distinguishes between graphs that the WL test also distinguishes. The following lemma inspired by lemma 2 in the article by Xu et al. formalizes this connection by showing how the representational power of message-passing GNNs (MP-GNNs) relates directly to that of the WL algorithm:

Lemma 1. *Let G_1 and G_2 be non-isomorphic graphs. If a graph neural network consists of K layers with the following update structure:*

$$h_u^{(k+1)} = UPDATE^{(k)}(h_u^{(k)}, AGGREGATE(\{h_v^{(k)}, \forall v \in N(u)\})) \qquad (11)$$

where $AGGREGATE$ is a differentiable permutation invariant function and UPDATE is a differentiable function. If this GNN maps G_1 and G_2 to different embeddings, the Weisfeiler-Lehman isomorphism test also decides G_1 and G_2 are not isomorphic.

In other words, the lemma states that a MP-GNN is at most as powerful as the WL-test in distinguishing different graphs.

The proof shows the contrapositive: that if the WL-test gives the same graph label to two graphs, the MP-GNN has to as well. At the node level, if WL node labels $l_v^{(i)} = l_u^{(i)}$ for nodes u, v in iteration i, then it is guaranteed that the MP-GNN node labels will be $h_v^{(i)} = h_u^{(i)}$. This is trivially true for iteration $i = 0$ since the input to the MP-GNN and WL-test is identical. Assuming that it holds for iteration j and that $l_v^{(j+1)} = l_u^{(j+1)}$, the authors prove that $h_v^{(j+1)} = h_u^{(j+1)}$ by referencing the properties of the AGGREGATE and UPDATE functions of the MP-GNN. This creates a mapping $h_v^{(i)} = \phi(l_v^{(i)})$ for any node v in either G_1 or G_2. This mapping implies that if the WL representations of nodes for the two graphs are the same, the MP-GNN embeddings must be too. Because the graph-level pooling function is permutation invariant, the graph representations/embeddings are the same as well.

The GIN Model

A natural question is to ask what, if anything, can make a GNN as powerful as the WL-test. Xu et al formulate an answer in a theorem that states that a MP-GNN will map any two graphs that the WL-test deems to be non-isomorphic to different embeddings if the AGGREGATE and UPDATE functions are injective over multisets of node embeddings. The authors also specify that in cases where learning on graphs is desired, the pooling used to produce a graph-level embedding should also be injective. The proof of the theorem creates a mapping between the new GNN and the WL update functions using the injectivity of the new AGGREGATE and UPDATE functions in the GNN as well as the injectivity of the hash function that updates the node labels in the WL-test. The proof that this mapping always holds throughout iterations of the algorithm is done by induction. Because the mapping is a composition of injective functions, it is itself injective. Therefore, when the WL algorithm maps graphs to distinct outputs, the new GNN will too, due to the injectivity of the mapping.

The authors create a model to realize their theoretical findings of a model that is able to match the expressive power of the WL test. They call it the Graph Isomorphism Network (GIN). The update rule of this model at layer k for node v is:

$$h_v^{(k)} = \mathrm{MLP}^{(k)}((1+\epsilon)^{(k)} \cdot h_v^{(k-1)} + \sum_{u \in N(v)} h_v^{(k-1)}), \tag{12}$$

where ϵ is a learnable or fixed scalar hyperparameter and MLP is a multilayer perceptron. A bit of background has to be mentioned to see how Xu et al got to this representation of the update rule for their model.

The first thing the authors set out to do is find an injective aggregation function over multisets, like the multiset of node features of the neighborhood of a target node. Mean and max are not injective aggregation functions over multisets but the authors find that sum can be. In a lemma, they state that there exists a function $f : \mathcal{X} \to \mathbb{R}^n$ such that $h(X) = \sum_{x \in X} f(x)$ is unique for each multiset $X \subset \mathcal{X}$ of bounded size, where $\mathcal{X}$ is countable. Moreover, any multiset function g can be written as $g(X) = \phi(\sum_{x \in X} f(x))$ for some function ϕ. This decomposition shows that summation followed by a learned function is sufficient to universally approximate any function over multisets.

To extend this expressivity to cases where node features are aggregated along with a target node's own features, the authors introduce the following function in a corollary:

$$h(c, X) = (1+\epsilon) \cdot f(c) + \sum_{x \in X} f(x), \tag{13}$$

where c is the is the feature of the central node and X is a multiset of its neighbors' features. The corollary states that for infinitely many choices of ϵ, including all irrational numbers, this function is injective. As with the lemma, any function g over such pairs can be expressed via the composition:

$$g(c, X) = \phi((1 + \epsilon) \cdot f(c) + \sum_{x \in X} f(x)). \tag{14}$$

The GIN model is created from this corollary. Due to the Universal Approximation Theorem, the MLP in the GIN update rule is used to learn f and ϕ in equation 14.

Beyond 1-WL

The comparisons between GNNs and the WL test that have been made thus far in this section have regarded the 1-WL test, where only individual nodes and their neighbors are considered. In the more powerful k-WL tests, subgraphs of size k are also considered. The following section will explore GNN models that surpass the 1-WL test and may instead have experssivity comparable to the k-WL test for some $k \geq 2$.

k-GNN

Morris et al. propose the k-dimensional Graph Neural Network (k-GNN) that generalizes the MP-GNN model by allowing it to pass information between sets of nodes of size k. The authors define the neighborhood of a k-set of nodes $s = \{s_1, s_2, ..., s_k\} \in [V(G)]^k$ to be:

$$N(s) = \{t \in [V(G)]^k, |s \cap t| = k - 1\}. \tag{15}$$

The k-GNN update rule looks much like the one for MP-GNNs, where information is aggregated from a node's neighborhood (or k-set of node's neighborhood $N(s)$ here). The authors prove that there exists some weights for the update rule such that k-GNNs are as powerful in graph discrimination as the k-WL test.

A major downside of the k-GNN model is its computational complexity. It is infeasible to implement or practically apply. Therefore, work has been put into finding other ways to create GNNs with the same distinguishing power as the k-WL test.

Short Practical Application

As an application of the research done to write the above, I have created Python programs for both the WL test and a very elementary MPGNN. I use the MUTAG dataset to apply my programs. This dataset regards molecules and whether or not they have a certain property. This property decides which class each graph (molecule) belongs to. On top of the WL test, I build a kernel for comparing graphs in MUTAG. Then, I use this kernel as an input to a SVM classifier that predicts which class the graphs in the dataset belong to. This procedure assesses how well the WL test distinguishes graphs, much like the test's original purpose isomorphism testing. I program the MPGNN to do graph classification using the following formula for layers:

$$h_v^{(t)} = \text{UPDATE}^{(t)}(h_v^{(t-1)}, \text{AGGREGATE}^{(t)}(\{h_u^{(t-1)}, \forall u \in N(v)\}), \tag{16}$$

and sum graph pooling. The test accuracy of the WL test based learning algorithm is 79% and the test accuracy of the MPGNN is 71%. This result is aligned with the theory detailed above: that such a basic MPGNN is at most as powerful in distinguishing graphs as the WL test.

Resources

- Main Book: Graph Representation Learning by William L. Hamilton
- Secondary Book: Geometric Deep Learning by Bronstein
- Xu et al. (2018): Introduced GIN and proved equivalence between 1-WL and message-passing GNNs.
- Morris et al. (2018): GNNs as 1-WL approximators, formalization of their expressive power.
- Murphy et al. (2019): Introduced Relational Pooling (RP)—a way to go beyond WL.
- Maron et al. (2019): Study of invariant/equivariant linear layers; deepens the understanding of graph symmetries and their expressive constraints.