Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks

Christopher Morris,[1] **Martin Ritzert,**[2] **Matthias Fey,**[1] **William L. Hamilton,**[3]
Jan Eric Lenssen,[1] **Gaurav Rattan,**[2] **Martin Grohe**[2]
[1]TU Dortmund University
[2]RWTH Aachen University
[3]McGill University and MILA
{christopher.morris, matthias.fey, janeric.lenssen}@tu-dortmund.de,
{ritzert, rattan, grohe}@informatik.rwth-aachen.de,
wlh@cs.mcgill.ca

Abstract

In recent years, graph neural networks (GNNs) have emerged as a powerful neural architecture to learn vector representations of nodes and graphs in a supervised, end-to-end fashion. Up to now, GNNs have only been evaluated empirically—showing promising results. The following work investigates GNNs from a theoretical point of view and relates them to the 1-dimensional Weisfeiler-Leman graph isomorphism heuristic (1-WL). We show that GNNs have the same expressiveness as the 1-WL in terms of distinguishing non-isomorphic (sub-)graphs. Hence, both algorithms also have the same shortcomings. Based on this, we propose a generalization of GNNs, so-called k-dimensional GNNs (k-GNNs), which can take higher-order graph structures at multiple scales into account. These higher-order structures play an essential role in the characterization of social networks and molecule graphs. Our experimental evaluation confirms our theoretical findings as well as confirms that higher-order information is useful in the task of graph classification and regression.

Introduction

Graph-structured data is ubiquitous across application domains ranging from chemo- and bioinformatics to image and social network analysis. To develop successful machine learning models in these domains, we need techniques that can exploit the rich information inherent in graph structure, as well as the feature information contained within a graph's nodes and edges. In recent years, numerous approaches have been proposed for machine learning graphs—most notably, approaches based on graph kernels (Vishwanathan et al. 2010) or, alternatively, using graph neural network algorithms (Hamilton, Ying, and Leskovec 2017b).

Kernel approaches typically fix a set of features in advance—e.g., indicator features over subgraph structures or features of local node neighborhoods. For example, one of the most successful kernel approaches, the *Weisfeiler-Lehman subtree kernel* (Shervashidze et al. 2011), which is based on the 1-dimensional Weisfeiler-Leman graph isomorphism heuristic (Grohe 2017, pp. 79 ff.), generates node features through an iterative relabeling, or *coloring*, scheme: First, all nodes are assigned a common initial color; the algorithm then iteratively recolors a node by aggregating over the multiset of colors in its neighborhood, and the final feature representation of a graph is the histogram of the resulting node colors. By iteratively aggregating over local node neighborhoods in this way, the WL subtree kernel is able to effectively summarize the neighborhood substructures present in a graph. However, while powerful, the WL subtree kernel—like other kernel methods—is limited because this feature construction scheme is fixed (i.e., it does not adapt to the given data distribution). Moreover, this approach—like the majority of kernel methods—focuses only on the graph structure and cannot interpret continuous node and edge labels, such as real-valued vectors which play an important role in applications such as bio- and chemoinformatics.

Graph neural networks (GNNs) have emerged as a machine learning framework addressing the above challenges. Standard GNNs can be viewed as a neural version of the 1-WL algorithm, where colors are replaced by continuous feature vectors and neural networks are used to aggregate over node neighborhoods (Hamilton, Ying, and Leskovec 2017a; Kipf and Welling 2017). In effect, the GNN framework can be viewed as implementing a continuous form of graph-based "message passing", where local neighborhood information is aggregated and passed on to the neighbors (Gilmer et al. 2017). By deploying a trainable neural network to aggregate information in local node neighborhoods, GNNs can be trained in an end-to-end fashion together with the parameters of the classification or regression algorithm, possibly allowing for greater adaptability and better generalization compared to the kernel counterpart of the classical 1-WL algorithm.

Up to now, the evaluation and analysis of GNNs has been largely empirical, showing promising results compared to kernel approaches, see, e.g., (Ying et al. 2018b). However, it remains unclear how GNNs are actually encoding graph structure information into their vector representations, and

whether there are theoretical advantages of GNNs compared to kernel based approaches.

Present Work. We offer a theoretical exploration of the relationship between GNNs and kernels that are based on the 1-WL algorithm. We show that GNNs cannot be more powerful than the 1-WL in terms of distinguishing non-isomorphic (sub-)graphs, e.g., the properties of subgraphs around each node. This result holds for a broad class of GNN architectures and all possible choices of parameters for them. On the positive side, we show that given the right parameter initialization GNNs have the same expressiveness as the 1-WL algorithm, completing the equivalence. Since the power of the 1-WL has been completely characterized, see, e.g., (Arvind et al. 2015; Kiefer, Schweitzer, and Selman 2015), we can transfer these results to the case of GNNs, showing that both approaches have the same shortcomings.

Going further, we leverage these theoretical relationships to propose a generalization of GNNs, called k-GNNs, which are neural architectures based on the k-dimensional WL algorithm (k-WL), which are strictly more powerful than GNNs. The key insight in these higher-dimensional variants is that they perform message passing directly between subgraph structures, rather than individual nodes. This higher-order form of message passing can capture structural information that is not visible at the node-level.

Graph kernels based on the k-WL have been proposed in the past (Morris, Kersting, and Mutzel 2017). However, a key advantage of implementing higher-order message passing in GNNs—which we demonstrate here—is that we can design hierarchical variants of k-GNNs, which combine graph representations learned at different granularities in an end-to-end trainable framework. Concretely, in the presented hierarchical approach the initial messages in a k-GNN are based on the output of lower-dimensional k'-GNN (with $k' < k$), which allows the model to effectively capture graph structures of varying granularity. Many real-world graphs inherit a hierarchical structure—e.g., in a social network we must model both the ego-networks around individual nodes, as well as the coarse-grained relationships between entire communities, see, e.g., (Newman 2003)—and our experimental results demonstrate that these hierarchical k-GNNs are able to consistently outperform traditional GNNs on a variety of graph classification and regression tasks. Across twelve graph regression tasks from the QM9 benchmark, we find that our hierarchical model reduces the mean absolute error by 54.45% on average. For graph classification, we find that our hierarchical models leads to slight performance gains.

Key Contributions. Our key contributions are summarized as follows:

1. We show that GNNs are not more powerful than the 1-WL in terms of distinguishing non-isomorphic (sub-)graphs. Moreover, we show that, assuming a suitable parameter initialization, GNNs have the same power as the 1-WL.
2. We propose k-GNNs, which are strictly more powerful than GNNs. Moreover, we propose a hierarchical version of k-GNNs, so-called 1-k-GNNs, which are able to work with the fine- and coarse-grained structures of a given graph, and relationships between those.
3. Our theoretical findings are backed-up by an experimental study, showing that higher-order graph properties are important for successful graph classification and regression.

Related Work

Our study builds upon a wealth of work at the intersection of supervised learning on graphs, kernel methods, and graph neural networks.

Historically, kernel methods—which implicitly or explicitly map graphs to elements of a Hilbert space—have been the dominant approach for supervised learning on graphs. Important early work in this area includes random-walk based kernels (Gärtner, Flach, and Wrobel 2003; Kashima, Tsuda, and Inokuchi 2003)) and kernels based on shortest paths (Borgwardt and Kriegel 2005). More recently, developments in graph kernels have emphasized scalability, focusing on techniques that bypass expensive Gram matrix computations by using explicit feature maps. Prominent examples of this trend include kernels based on graphlet counting (Shervashidze et al. 2009), and, most notably, the Weisfeiler-Lehman subtree kernel (Shervashidze et al. 2011) as well as its higher-order variants (Morris, Kersting, and Mutzel 2017). Graphlet and Weisfeiler-Leman kernels have been successfully employed within frameworks for smoothed and deep graph kernels (Yanardag and Vishwanathan 2015a; 2015b). Recent works focus on assignment-based approaches (Kriege, Giscard, and Wilson 2016; Nikolentzos, Meladianos, and Vazirgiannis 2017; Johansson and Dubhashi 2015), spectral approaches (Kondor and Pan 2016), and graph decomposition approaches (Nikolentzos et al. 2018). Graph kernels were dominant in graph classification for several years, leading to new state-of-the-art results on many classification tasks. However, they are limited by the fact that they cannot effectively adapt their feature representations to a given data distribution, since they generally rely on a fixed set of features. More recently, a number of approaches to graph classification based upon neural networks have been proposed. Most of the neural approaches fit into the graph neural network framework proposed by (Gilmer et al. 2017). Notable instances of this model include *Neural Fingerprints* (Duvenaud et al. 2015), *Gated Graph Neural Networks* (Li et al. 2016), *GraphSAGE* (Hamilton, Ying, and Leskovec 2017a), *SplineCNN* (Fey et al. 2018), and the spectral approaches proposed in (Bruna et al. 2014; Defferrard, X., and Vandergheynst 2016; Kipf and Welling 2017)—all of which descend from early work in (Merkwirth and Lengauer 2005) and (Scarselli et al. 2009b). Recent extensions and improvements to the GNN framework include approaches to incorporate different local structures around subgraphs (Xu et al. 2018) and novel techniques for pooling node representations in order perform graph classification (Zhang et al. 2018; Ying et al. 2018b). GNNs have achieved state-of-the-art performance on several graph classification benchmarks in recent years, see, e.g., (Ying et al. 2018b)—as well as applications such as protein-protein interaction prediction (Fout et al. 2017), recommender systems (Ying et al. 2018a), and the analysis of quantum interactions in molecules (Schütt et al. 2017). A survey of recent advancements in GNN techniques can be found in (Hamilton, Ying, and Leskovec 2017b).

Up to this point (and despite their empirical success) there has been very little theoretical work on GNNs—with the no-

table exceptions of Li et al.'s (Li, Han, and Wu 2018) work connecting GNNs to a special form Laplacian smoothing and Lei et al.'s (Lei et al. 2017) work showing that the feature maps generated by GNNs lie in the same Hilbert space as some popular graph kernels. Moreover, Scarselli et al. (Scarselli et al. 2009a) investigates the approximation capabilities of GNNs.

Preliminaries

We start by fixing notation, and then outline the Weisfeiler-Leman algorithm and the standard graph neural network framework.

Notation and Background

A *graph* G is a pair (V, E) with a finite set of *nodes* V and a set of *edges* $E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$. We denote the set of nodes and the set of edges of G by $V(G)$ and $E(G)$, respectively. For ease of notation we denote the edge $\{u, v\}$ in $E(G)$ by (u, v) or (v, u). Moreover, $N(v)$ denotes the *neighborhood* of v in $V(G)$, i.e., $N(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$. We say that two graphs G and H are *isomorphic* if there exists an edge preserving bijection $\varphi : V(G) \to V(H)$, i.e., (u, v) is in $E(G)$ if and only if $(\varphi(u), \varphi(v))$ is in $E(H)$. We write $G \simeq H$ and call the equivalence classes induced by $\simeq$ *isomorphism types*. Let $S \subseteq V(G)$ then $G[S] = (S, E_S)$ is the *subgraph induced* by S with $E_S = \{(u, v) \in E(G) \mid u, v \in S\}$. A *node coloring* is a function $V(G) \to \Sigma$ with arbitrary codomain Σ. Then a *node colored* or *labeled graph* (G, l) is a graph G endowed with a node coloring $l \colon V(G) \to \Sigma$. We say that $l(v)$ is a *label* or *color* of $v \in V(G)$. We say that a node coloring c *refines* a node coloring d, written $c \sqsubseteq d$, if $c(v) = c(w)$ implies $d(v) = d(w)$ for every v, w in $V(G)$. Two colorings are *equivalent* if $c \sqsubseteq d$ and $d \sqsubseteq c$, and we write $c \equiv d$. A *color class* $Q \subseteq V(G)$ of a node coloring c is a maximal set of nodes with $c(v) = c(w)$ for every v, w in Q. Moreover, let $[1\colon n] = \{1, \ldots, n\} \subset \mathbb{N}$ for $n > 1$, let S be a set then the set of *k-sets* $[S]^k = \{U \subseteq S \mid |U| = k\}$ for $k \geq 2$, which is the set of all subsets with cardinality k, and let $\{\!\{\ldots\}\!\}$ denote a multiset.

Weisfeiler-Leman Algorithm

We now describe the 1-WL algorithm for labeled graphs. Let (G, l) be a labeled graph. In each iteration, $t \geq 0$, the 1-WL computes a node coloring $c_l^{(t)} \colon V(G) \to \Sigma$, which depends on the coloring from the previous iteration. In iteration 0, we set $c_l^{(0)} = l$. Now in iteration $t > 0$, we set

$$c_l^{(t)}(v) = \textsc{hash}\Big(\big(c_l^{(t-1)}(v), \{\!\{c_l^{(t-1)}(u) \mid u \in N(v)\}\!\}\big)\Big) \quad (1)$$

where HASH bijectively maps the above pair to a unique value in Σ, which has not been used in previous iterations. To test two graph G and H for isomorphism, we run the above algorithm in "parallel" on both graphs. Now if the two graphs have a different number of nodes colored σ in Σ, the 1-WL concludes that the graphs are not isomorphic. Moreover, if the number of colors between two iterations does not change, i.e., the cardinalities of the images of $c_l^{(t-1)}$ and $c_l^{(t)}$ are equal, the algorithm terminates. Termination is guaranteed after at most $\max\{|V(G)|, |V(H)|\}$ iterations. It is easy to see that the algorithm is not able to distinguish all non-isomorphic graphs, e.g., see (Cai, Fürer, and Immerman 1992). Nonetheless, it is a powerful heuristic, which can successfully test isomorphism for a broad class of graphs (Babai and Kucera 1979).

The k-dimensional Weisfeiler-Leman algorithm (k-WL), for $k \geq 2$, is a generalization of the 1-WL which colors tuples from $V(G)^k$ instead of nodes. That is, the algorithm computes a coloring $c_{l,k}^{(t)} \colon V(G)^k \to \Sigma$. In order to describe the algorithm, we define the j-th neighborhood

$$N_j(s) = \{(s_1, \ldots, s_{j-1}, r, s_{j+1}, \ldots, s_k) \mid r \in V(G)\} \quad (2)$$

of a k-tuple $s = (s_1, \ldots, s_k)$ in $V(G)^k$. That is, the j-th neighborhood $N_j(t)$ of s is obtained by replacing the j-th component of s by every node from $V(G)$. In iteration 0, the algorithm labels each k-tuple with its *atomic type*, i.e., two k-tuples s and s' in $V(G)^k$ get the same color if the map $s_i \mapsto s'_i$ induces a (labeled) isomorphism between the subgraphs induced from the nodes from s and s', respectively. For iteration $t > 0$, we define

$$C_j^{(t)}(s) = \textsc{hash}\big(\{\!\{c_{l,k}^{(t-1)}(s') \mid s' \in N_j(s)\}\!\}\big), \quad (3)$$

and set

$$c_{k,l}^{(t)}(s) = \textsc{hash}\Big(\big(c_{k,l}^{(t-1)}(s), \big(C_1^{(t)}(s), \ldots, C_k^{(t)}(s)\big)\big)\Big). \quad (4)$$

Hence, two tuples s and s' with $c_{k,l}^{(t-1)}(s) = c_{k,l}^{(t-1)}(s')$ get different colors in iteration t if there exists j in $[1\colon k]$ such that the number of j-neighbors of s and s', respectively, colored with a certain color is different. The algorithm then proceeds analogously to the 1-WL. By increasing k, the algorithm gets more powerful in terms of distinguishing non-isomorphic graphs, i.e., for each $k \geq 2$, there are non-isomorphic graphs which can be distinguished by the $(k+1)$-WL but not by the k-WL (Cai, Fürer, and Immerman 1992). We note here that the above variant is not equal to the *folklore* variant of k-WL described in (Cai, Fürer, and Immerman 1992), which differs slightly in its update rule. However, it holds that the k-WL using Equation (4) is as powerful as the folklore $(k-1)$-WL (Grohe and Otto 2015).

WL Kernels. After running the WL algorithm, the concatenation of the histogram of colors in each iteration can be used as a feature vector in a kernel computation. Specifically, in the histogram for every color σ in Σ there is an entry containing the number of nodes or k-tuples that are colored with σ.

Graph Neural Networks

Let (G, l) be a labeled graph with an initial node coloring $f^{(0)} : V(G) \to \mathbb{R}^{1\times d}$ that is *consistent* with l. This means that each node v is annotated with a feature $f^{(0)}(v)$ in $\mathbb{R}^{1\times d}$ such that $f^{(0)}(u) = f^{(0)}(v)$ if and only if $l(u) = l(v)$. Alternatively, $f^{(0)}(v)$ can be an arbitrary real-valued feature vector associated with v. Examples include continuous atomic properties in chemoinformatic applications where nodes correspond to atoms, or vector representations of text in social network applications. A GNN model consists of a stack of neural network layers, where each layer aggregates local neighborhood information, i.e., features of neighbors, around each node and then passes this aggregated information on to the next layer.

A basic GNN model can be implemented as follows (Hamilton, Ying, and Leskovec 2017b). In each layer $t > 0$, we compute a new feature

$$f^{(t)}(v) = \sigma\Big(f^{(t-1)}(v)\cdot W_1^{(t)} + \sum_{w\in N(v)} f^{(t-1)}(w)\cdot W_2^{(t)}\Big) \quad (5)$$

in $\mathbb{R}^{1\times e}$ for v, where $W_1^{(t)}$ and $W_2^{(t)}$ are parameter matrices from $\mathbb{R}^{d\times e}$, and σ denotes a component-wise non-linear function, e.g., a sigmoid or a ReLU.[1]

Following (Gilmer et al. 2017), one may also replace the sum defined over the neighborhood in the above equation by a permutation-invariant, differentiable function, and one may substitute the outer sum, e.g., by a column-wise vector concatenation or LSTM-style update step. Thus, in full generality a new feature $f^{(t)}(v)$ is computed as

$$f_{\text{merge}}^{W_1}\Big(f^{(t-1)}(v), f_{\text{aggr}}^{W_2}\big(\{\!\{f^{(t-1)}(w) \mid w \in N(v)\}\!\}\big)\Big), \quad (6)$$

where $f_{\text{aggr}}^{W_1}$ aggregates over the set of neighborhood features and $f_{\text{merge}}^{W_2}$ merges the node's representations from step $(t-1)$ with the computed neighborhood features. Both $f_{\text{aggr}}^{W_1}$ and $f_{\text{merge}}^{W_2}$ may be arbitrary differentiable, permutation-invariant functions (e.g., neural networks), and, by analogy to Equation 5, we denote their parameters as W_1 and W_2, respectively. In the rest of this paper, we refer to neural architectures implementing Equation (6) as 1-*dimensional GNN architectures* (1-GNNs).

A vector representation f_{GNN} over the whole graph can be computed by summing over the vector representations computed for all nodes, i.e.,

$$f_{\text{GNN}}(G) = \sum_{v\in V(G)} f^{(T)}(v),$$

where $T > 0$ denotes the last layer. More refined approaches use differential pooling operators based on sorting (Zhang et al. 2018) and soft assignments (Ying et al. 2018b).

In order to adapt the parameters W_1 and W_2 of Equations (5) and (6), to a given data distribution, they are optimized in an end-to-end fashion (usually via stochastic gradient descent) together with the parameters of a neural network used for classification or regression.

Relationship Between 1-WL and 1-GNNs

In the following we explore the relationship between the 1-WL and 1-GNNs. Let (G, l) be a labeled graph, and let $\mathbf{W}^{(t)} = \big(W_1^{(t')}, W_2^{(t')}\big)_{t'\le t}$ denote the GNN parameters given by Equation (5) or Equation (6) up to iteration t. We encode the initial labels $l(v)$ by vectors $f^{(0)}(v) \in \mathbb{R}^{1\times d}$, e.g., using a 1-hot encoding.

Our first theoretical result shows that the 1-GNN architectures do not have more power in terms of distinguishing between non-isomorphic (sub-)graphs than the 1-WL algorithm. More formally, let $f_{\text{merge}}^{W_1}$ and $f_{\text{aggr}}^{W_2}$ be any two functions chosen in (6). For every encoding of the labels $l(v)$ as vectors $f^{(0)}(v)$, and for every choice of $\mathbf{W}^{(t)}$, we have that the coloring $c_l^{(t)}$ of 1-WL always refines the coloring $f^{(t)}$ induced by a 1-GNN parameterized by $\mathbf{W}^{(t)}$.

Theorem 1. Let (G, l) be a labeled graph. Then for all $t \ge 0$ and for all choices of initial colorings $f^{(0)}$ consistent with l, and weights $\mathbf{W}^{(t)}$,

$$c_l^{(t)} \sqsubseteq f^{(t)}\,.$$

Our second result states that there exist a sequence of parameter matrices $\mathbf{W}^{(t)}$ such that 1-GNNs have exactly the same power in terms of distinguishing non-isomorphic (sub-)graphs as the 1-WL algorithm. This even holds for the simple architecture (5), provided we choose the encoding of the initial labeling l in such a way that different labels are encoded by linearly independent vectors.

Theorem 2. Let (G, l) be a labeled graph. Then for all $t \ge 0$ there exists a sequence of weights $\mathbf{W}^{(t)}$, and a 1-GNN architecture such that

$$c_l^{(t)} \equiv f^{(t)}\,.$$

Hence, in the light of the above results, 1-GNNs may viewed as an extension of the 1-WL which in principle have the same power but are more flexible in their ability to adapt to the learning task at hand and are able to handle continuous node features.

Shortcomings of Both Approaches

The power of 1-WL has been completely characterized, see, e.g., (Arvind et al. 2015). Hence, by using Theorems 1 and 2, this characterization is also applicable to 1-GNNs. On the other hand, 1-GNNs have the same shortcomings as the 1-WL. For example, both methods will give the same color to every node in a graph consisting of a triangle and a 4-cycle, although vertices from the triangle and the vertices from the 4-cycle are clearly different. Moreover, they are not capable of capturing simple graph theoretic properties, e.g., triangle counts, which are an important measure in social network analysis (Milo et al. 2002; Newman 2003).

k-dimensional Graph Neural Networks

In the following, we propose a generalization of 1-GNNs, so-called k-GNNs, which are based on the k-WL. Due to scalability and limited GPU memory, we consider a set-based version of the k-WL. For a given k, we consider all k-element subsets $[V(G)]^k$ over $V(G)$. Let $s = \{s_1, \ldots, s_k\}$ be a k-set in $[V(G)]^k$, then we define the *neighborhood* of s as

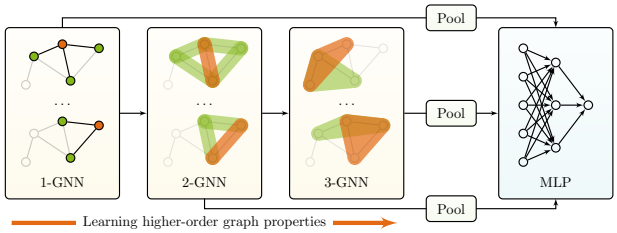
$$N(s) = \{t \in [V(G)]^k \mid |s \cap t| = k - 1\}\,.$$

所有跟标的的k集只有一个节点不同的其他k集，作为这个k集的邻居

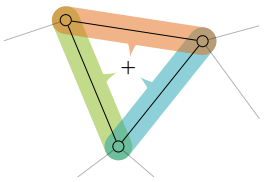
The *local neighborhood* $N_L(s)$ consists of all $t \in N(s)$ such that $(v, w) \in E(G)$ for the unique $v \in s \setminus t$ and the unique $w \in t \setminus s$. The *global neighborhood* $N_G(s)$ then is defined as $N(s) \setminus N_L(s)$.[2]

[1] For clarity of presentation we omit biases.

[2] Note that the definition of the local neighborhood is different from the the one defined in (Morris, Kersting, and Mutzel 2017) which is a superset of our definition. Our computations therefore involve sparser graphs.



(a) Hierarchical 1-2-3-GNN network architecture



(b) Pooling from 2- to 3-GNN.

Figure 1: Illustration of the proposed hierarchical variant of the k-GNN layer. For each subgraph S on k nodes a feature f is learned, which is initialized with the learned features of all $(k-1)$-element subgraphs of S. Hence, a hierarchical representation of the input graph is learned.

The set based k-WL works analogously to the k-WL, i.e., it computes a coloring $c^{(t)}_{s,k,l} : [V(G)]^k \to \Sigma$ as in Equation (1) based on the above neighborhood. Initially, $c^{(0)}_{s,k,l}$ colors each element s in $[V(G)]^k$ with the isomorphism type of $G[s]$.

Let (G, l) be a labeled graph. In each k-GNN layer $t \geq 0$, we compute a feature vector $f^{(t)}_k(s)$ for each k-set s in $[V(G)]^k$. For $t = 0$, we set $f^{(0)}_k(s)$ to $f^{\text{iso}}(s)$, a one-hot encoding of the isomorphism type of $G[s]$ labeled by l. In each layer $t > 0$, we compute new features by

$$f^{(t)}_k(s) = \sigma\Big(f^{(t-1)}_k(s) \cdot W^{(t)}_1 + \sum_{u \in N_L(s) \cup N_G(s)} f^{(t-1)}_k(u) \cdot W^{(t)}_2\Big).$$

Moreover, one could split the sum into two sums ranging over $N_L(s)$ and $N_G(s)$ respectively, using distinct parameter matrices to enable the model to learn the importance of local and global neighborhoods. To scale k-GNNs to larger datasets and to prevent overfitting, we propose *local* k-GNNs, where we omit the global neighborhood of s, i.e.,

$$f^{(t)}_{k,\text{L}}(s) = \sigma\Big(f^{(t-1)}_{k,\text{L}}(s) \cdot W^{(t)}_1 + \sum_{u \in N_L(s)} f^{(t-1)}_{k,\text{L}}(u) \cdot W^{(t)}_2\Big).$$

The running time for evaluation of the above depends on $|V|$, k and the sparsity of the graph (each iteration can be bounded by the number of subsets of size k times the maximum degree). Note that we can scale our method to larger datasets by using sampling strategies introduced in, e.g., (Morris, Kersting, and Mutzel 2017; Hamilton, Ying, and Leskovec 2017a). We can now lift the results of the previous section to the k-dimensional case.

Proposition 3. Let (G, l) be a labeled graph and let $k \geq 2$. Then for all $t \geq 0$, for all choices of initial colorings $f^{(0)}_k$ consistent with l and for all weights $\mathbf{W}^{(t)}$,

$$c^{(t)}_{s,k,l} \sqsubseteq f^{(t)}_k.$$

Again the second result states that there exists a suitable initialization of the parameter matrices $\mathbf{W}^{(t)}$ such that k-GNNs have exactly the same power in terms of distinguishing non-isomorphic (sub-)graphs as the set-based k-WL.

Proposition 4. Let (G, l) be a labeled graph and let $k \geq 2$. Then for all $t \geq 0$ there exists a sequence of weights $\mathbf{W}^{(t)}$, and a k-GNN architecture such that

$$c^{(t)}_{s,k,l} \equiv f^{(t)}_k.$$

Hierarchical Variant

One key benefit of the end-to-end trainable k-GNN framework—compared to the discrete k-WL algorithm—is that we can hierarchically combine representations learned at different granularities. Concretely, rather than simply using one-hot indicator vectors as initial feature inputs in a k-GNN, we propose a *hierarchical* variant of k-GNN that uses the features learned by a $(k-1)$-dimensional GNN, in addition to the (labeled) isomorphism type, as the initial features, i.e.,

$$f^{(0)}_k(s) = \sigma\Big(\Big[f^{\text{iso}}(s), \sum_{u \subset s} f^{(T_{k-1})}_{k-1}(u)\Big] \cdot W_{k-1}\Big),$$

for some $T_{k-1} > 0$, where W_{k-1} is a matrix of appropriate size, and square brackets denote matrix concatenation.

Hence, the features are recursively learned from dimensions 1 to k in an end-to-end fashion. This hierarchical model also satisfies Propositions 3 and 4, so its representational capacity is theoretically equivalent to a standard k-GNN (in terms of its relationship to k-WL). Nonetheless, hierarchy is a natural inductive bias for graph modeling, since many real-world graphs incorporate hierarchical structure, so we expect this hierarchical formulation to offer empirical utility.

Experimental Study

In the following, we want to investigate potential benefits of GNNs over graph kernels as well as the benefits of our proposed k-GNN architectures over 1-GNN architectures. More precisely, we address the following questions:

Q1 How do the (hierarchical) k-GNNs perform in comparison to state-of-the-art graph kernels?

Q2 How do the (hierarchical) k-GNNs perform in comparison to the 1-GNN in graph classification and regression tasks?

Q3 How much (if any) improvement is provided by optimizing the parameters of the GNN aggregation function, compared to just using random GNN parameters while optimizing the parameters of the downstream classification/regression algorithm?

Datasets

To compare our k-GNN architectures to kernel approaches we use well-established benchmark datasets from the graph kernel literature (Kersting et al. 2016). The nodes of each graph in these dataset is annotated with (discrete) labels or no labels.

Table 1: Classification accuracies in percent on various graph benchmark datasets.

	Method	Dataset						
		PRO	IMDB-BIN	IMDB-MUL	PTC-FM	NCI1	MUTAG	PTC-MR
Kernel	GRAPHLET	72.9	59.4	40.8	58.3	72.1	87.7	54.7
	SHORTEST-PATH	**76.4**	59.2	40.5	62.1	74.5	81.7	58.9
	1-WL	73.8	72.5	**51.5**	62.9	83.1	78.3	61.3
	2-WL	75.2	72.6	50.6	**64.7**	77.0	77.0	61.9
	3-WL	74.7	73.5	49.7	61.5	83.1	83.2	62.5
	WL-OA	75.3	73.1	50.4	62.7	**86.1**	84.5	**63.6**
GNN	DCNN	61.3	49.1	33.5	—	62.6	67.0	56.6
	PATCHYSAN	75.9	71.0	45.2	—	78.6	**92.6**	60.0
	DGCNN	75.5	70.0	47.8	—	74.4	85.8	58.6
	1-GNN NO TUNING	70.7	69.4	47.3	59.0	58.6	82.7	51.2
	1-GNN	72.2	71.2	47.7	59.3	74.3	82.2	59.0
	1-2-3-GNN NO TUNING	75.9	70.3	48.8	60.0	67.4	84.4	59.3
	1-2-3-GNN	75.5	**74.2**	49.5	62.8	76.2	86.1	60.9

Table 2: Mean absolute errors on the QM9 dataset. The far-right column shows the improvement of the best k-GNN model in comparison to the 1-GNN baseline.

Target	Method						
	DTNN (Wu et al. 2018)	MPNN (Wu et al. 2018)	1-GNN	1-2-GNN	1-3-GNN	1-2-3-GNN	Gain
μ	**0.244**	0.358	0.493	0.493	**0.473**	0.476	4.0%
α	0.95	0.89	0.78	**0.27**	0.46	**0.27**	65.3%
$\varepsilon_{\text{HOMO}}$	0.00388	0.00541	**0.00321**	0.00331	0.00328	0.00337	–
$\varepsilon_{\text{LUMO}}$	0.00512	0.00623	0.00355	**0.00350**	0.00354	0.00351	1.4%
$\Delta\varepsilon$	0.0112	0.0066	0.0049	0.0047	**0.0046**	0.0048	6.1%
$\langle R^2 \rangle$	**17.0**	28.5	34.1	21.5	25.8	22.9	37.0%
ZPVE	0.00172	0.00216	0.00124	**0.00018**	0.00064	0.00019	85.5%
U_0	2.43	2.05	2.32	**0.0357**	0.6855	0.0427	98.5%
U	2.43	2.00	2.08	**0.107**	0.686	0.111	94.9%
H	2.43	2.02	2.23	0.070	0.794	**0.0419**	98.1%
G	2.43	2.02	1.94	0.140	0.587	**0.0469**	97.6%
C_{v}	0.27	0.42	0.27	0.0989	0.158	**0.0944**	65.0%

To demonstrate that our architectures scale to larger datasets and offer benefits on real-world applications, we conduct experiments on the QM9 dataset (Ramakrishnan et al. 2014; Ruddigkeit et al. 2012; Wu et al. 2018), which consists of 133 385 small molecules. The aim here is to perform regression on twelve targets representing energetic, electronic, geometric, and thermodynamic properties, which were computed using density functional theory.

Baselines

We use the following kernel and GNN methods as baselines for our experiments.

Kernel Baselines. We use the Graphlet kernel (Shervashidze et al. 2009), the shortest-path kernel (Borgwardt and Kriegel 2005), the Weisfeiler-Lehman subtree kernel (WL) (Shervashidze et al. 2011), the Weisfeiler-Lehman Optimal Assignment kernel (WL-OA) (Kriege, Giscard, and Wilson 2016), and the global-local k-WL (Morris, Kersting, and Mutzel 2017) with k in $\{2, 3\}$ as kernel baselines. For each kernel, we computed the normalized Gram matrix. We used the C-SVM implementation of LIBSVM (Chang and Lin 2011) to compute the classification accuracies using 10-fold cross validation. The parameter C was selected from $\{10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}\}$ by 10-fold cross validation on the training folds.

Neural Baselines. To compare GNNs to kernels we used the basic 1-GNN layer of Equation (5), DCNN (Wang et al. 2018), PatchySan (Niepert, Ahmed, and Kutzkov 2016), DGCNN (Zhang et al. 2018). For the QM9 dataset we used a 1-GNN layer similar to (Gilmer et al. 2017), where we replaced the inner sum of Equation (5) with a 2-layer MLP in order incorporate edge features (bond type and distance information). Moreover, we compare against the numbers provided in (Wu et al. 2018).

Model Configuration

We always used three layers for 1-GNN, and two layers for (local) 2-GNN and 3-GNN, all with a hidden-dimension size of 64. For the hierarchical variant we used architectures that use features computed by 1-GNN as initial features for the 2-GNN (1-2-GNN) and 3-GNN (1-3-GNN), respectively. Moreover, us-

ing the combination of the former we componentwise concatenated the computed features of the 1-2-GNN and the 1-3-GNN (1-2-3-GNN). For the final classification and regression steps, we used a three layer MLP, with binary cross entropy and mean squared error for the optimization, respectively. For classification we used a dropout layer with $p = 0.5$ after the first layer of the MLP. We applied global average pooling to generate a vector representation of the graph from the computed node features for each k. The resulting vectors are concatenated column-wise before feeding them into the MLP. Moreover, we used the Adam optimizer with an initial learning rate of 10^{-2} and applied an adaptive learning rate decay based on validation results to a minimum of 10^{-5}. We trained the classification networks for 100 epochs and the regression networks for 200 epochs.

Experimental Protocol

For the smaller datasets, which we use for comparison against the kernel methods, we performed a 10-fold cross validation where we randomly sampled 10% of each training fold to act as a validation set. For the QM9 dataset, we follow the dataset splits described in (Wu et al. 2018). We randomly sampled 10% of the examples for validation, another 10% for testing, and used the remaining for training. We used the same initial node features as described in (Gilmer et al. 2017). Moreover, in order to illustrate the benefits of our hierarchical k-GNN architecture, we did not use a complete graph, where edges are annotated with pairwise distances, as input. Instead, we only used pairwise Euclidean distances for connected nodes, computed from the provided node coordinates. The code was built upon the work of (Fey et al. 2018) and is provided at https://github.com/chrsmrrs/k-gnn.

Results and Discussion

In the following we answer questions **Q1** to **Q3**. Table 1 shows the results for comparison with the kernel methods on the graph classification benchmark datasets. Here, the hierarchical k-GNN is on par with the kernels despite the small dataset sizes (answering question **Q1**). We also find that the 1-2-3-GNN significantly outperforms the 1-GNN on all seven datasets (answering **Q2**), with the 1-GNN being the overall weakest method across all tasks.[3] We can further see that optimizing the parameters of the aggregation function only leads to slight performance gains on two out of three datasets, and that no optimization even achieves better results on the PROTEINS benchmark dataset (answering **Q3**). We contribute this effect to the one-hot encoded node labels, which allow the GNN to gather enough information out of the neighborhood of a node, even when this aggregation is not learned.

Table 2 shows the results for the QM9 dataset. On eleven out of twelve targets all of our hierarchical variants beat the 1-GNN baseline, providing further evidence for **Q2**. For example, on the target H we achieve a large improvement of 98.1% in MAE compared to the baseline. Moreover, on ten out of twelve datasets, the hierarchical k-GNNs beat the baselines from (Wu et al. 2018). However, the additional structural information extracted by the k-GNN layers does not serve all tasks equally, leading to huge differences in gains across the targets.

It should be noted that our k-GNN models have more parameters than the 1-GNN model, since we stack two additional GNN layers for each k. However, extending the 1-GNN model by additional layers to match the number of parameters of the k-GNN did not lead to better results in any experiment.

[3]Note that in very recent work, GNNs have shown superior results over kernels when using advanced pooling techniques (Ying et al. 2018b). Note that our layers can be combined with these pooling layers. However, we opted to use standard global pooling in order to compare a typical GNN implementation with standard off-the-shelf kernels.

Conclusion

We presented a theoretical investigation of GNNs, showing that a wide class of GNN architectures cannot be stronger than the 1-WL. On the positive side, we showed that, in principle, GNNs possess the same power in terms of distinguishing between non-isomorphic (sub-)graphs, while having the added benefit of adapting to the given data distribution. Based on this insight, we proposed k-GNNs which are a generalization of GNNs based on the k-WL. This new model is strictly stronger then GNNs in terms of distinguishing non-isomorphic (sub-)graphs and is capable of distinguishing more graph properties. Moreover, we devised a hierarchical variant of k-GNNs, which can exploit the hierarchical organization of most real-world graphs. Our experimental study shows that k-GNNs consistently outperform 1-GNNs and beat state-of-the-art neural architectures on large-scale molecule learning tasks. Future work includes designing task-specific k-GNNs, e.g., devising k-GNNs layers that exploit expert-knowledge in bio- and chemoinformatic settings.

Acknowledgments

This work is supported by the German research council (DFG) within the Research Training Group 2236 *UnRAVeL* and the Collaborative Research Center SFB 876, *Providing Information by Resource-Constrained Analysis*, projects A6 and B2.

References

Arvind, V.; Köbler, J.; Rattan, G.; and Verbitsky, O. 2015. On the power of color refinement. In *Symposium on Fundamentals of Computation Theory*, 339–350.

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

Borgwardt, K. M., and Kriegel, H.-P. 2005. Shortest-path kernels on graphs. In *ICDM*, 74–81.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2014. Spectral networks and deep locally connected networks on graphs. In *ICLR*.

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

Chang, C.-C., and Lin, C.-J. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2:27:1–27:27.

Defferrard, M.; X., B.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 3844–3852.

Duvenaud, D. K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, 2224–2232.

Fey, M.; Lenssen, J. E.; Weichert, F.; and Müller, H. 2018. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *CVPR*.

Fout, A.; Byrd, J.; Shariat, B.; and Ben-Hur, A. 2017. Protein interface prediction using graph convolutional networks. In *NIPS*, 6533–6542.

Gärtner, T.; Flach, P.; and Wrobel, S. 2003. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*. 129–143.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *ICML*.

Grohe, M., and Otto, M. 2015. Pebble games and linear equations. *Journal of Symbolic Logic* 80(3):797–844.

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

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017a. Inductive representation learning on large graphs. In *NIPS*, 1025–1035.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017b. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin* 40(3):52–74.

Johansson, F. D., and Dubhashi, D. 2015. Learning with similarity functions on graphs using matchings of geometric embeddings. In *KDD*, 467–476.

Kashima, H.; Tsuda, K.; and Inokuchi, A. 2003. Marginalized kernels between labeled graphs. In *ICML*, 321–328.

Kersting, K.; Kriege, N. M.; Morris, C.; Mutzel, P.; and Neumann, M. 2016. Benchmark data sets for graph kernels.

Kiefer, S.; Schweitzer, P.; and Selman, E. 2015. Graphs identified by logics with counting. In *MFCS*, 319–330. Springer.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Kondor, R., and Pan, H. 2016. The multiscale laplacian graph kernel. In *NIPS*, 2982–2990.

Kriege, N. M.; Giscard, P.-L.; and Wilson, R. C. 2016. On valid optimal assignment kernels and applications to graph classification. In *NIPS*, 1615–1623.

Lei, T.; Jin, W.; Barzilay, R.; and Jaakkola, T. S. 2017. Deriving neural architectures from sequence and graph kernels. In *ICML*, 2024–2033.

Li, W.; Saidi, H.; Sanchez, H.; Schäf, M.; and Schweitzer, P. 2016. Detecting similar programs via the Weisfeiler-Leman graph kernel. In *International Conference on Software Reuse*, 315–330.

Li, Q.; Han, Z.; and Wu, X.-M. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 3538–3545.

Merkwirth, C., and Lengauer, T. 2005. Automatic generation of complementary descriptors with molecular graph networks. *Journal of Chemical Information and Modeling* 45(5):1159–1168.

Milo, R.; Shen-Orr, S.; Itzkovitz, S.; Kashtan, N.; Chklovskii, D.; and Alon, U. 2002. Network motifs: simple building blocks of complex networks. *Science* 298(5594):824–827.

Morris, C.; Kersting, K.; and Mutzel, P. 2017. Glocalized Weisfeiler-Lehman kernels: Global-local feature maps of graphs. In *ICDM*, 327–336.

Newman, M. E. J. 2003. The structure and function of complex networks. *SIAM review* 45(2):167–256.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *ICML*, 2014–2023.

Nikolentzos, G.; Meladianos, P.; Limnios, S.; and Vazirgiannis, M. 2018. A degeneracy framework for graph similarity. In *IJCAI*, 2595–2601.

Nikolentzos, G.; Meladianos, P.; and Vazirgiannis, M. 2017. Matching node embeddings for graph similarity. In *AAAI*, 2429–2435.

Ramakrishnan, R.; Dral, P., O.; Rupp, M.; and von Lilienfeld, O. A. 2014. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data* 1.

Ruddigkeit, L.; van Deursen, R.; Blum, L. C.; and Reymond, J.-L. 2012. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of Chemical Information and Modeling* 52 11:2864–75.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009a. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks* 20(1):81–102.

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

Schütt, K.; Kindermans, P. J.; Sauceda, H. E.; Chmiela, S.; Tkatchenko, A.; and Müller, K. R. 2017. SchNet: A continuous-filter convolutional neural network for modeling quantum interactions. In *NIPS*, 992–1002.

Shervashidze, N.; Vishwanathan, S. V. N.; Petri, T. H.; Mehlhorn, K.; and Borgwardt, K. M. 2009. Efficient graphlet kernels for large graph comparison. In *AISTATS*, 488–495.

Shervashidze, N.; Schweitzer, P.; van Leeuwen, E. J.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-Lehman graph kernels. *JMLR* 12:2539–2561.

Vishwanathan, S. V. N.; Schraudolph, N. N.; Kondor, R.; and Borgwardt, K. M. 2010. Graph kernels. *JMLR* 11:1201–1242.

Wang, Y.; Sun, Y.; Liu, Z.; Sarma, S. E.; Bronstein, M. M.; and Solomon, J. M. 2018. Dynamic graph CNN for learning on point clouds. *CoRR* abs/1801.07829.

Wu, Z.; Ramsundar, B.; Feinberg, E. N.; Gomes, J.; Geniesse, C.; Pappu, A. S.; Leswing, K.; and Pande, V. 2018. Moleculenet: a benchmark for molecular machine learning. *Chemical Science* 9:513–530.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018. Representation learning on graphs with jumping knowledge networks. In *ICML*, 5453–5462.

Yanardag, P., and Vishwanathan, S. V. N. 2015a. Deep graph kernels. In *KDD*, 1365–1374.

Yanardag, P., and Vishwanathan, S. V. N. 2015b. A structural smoothing framework for robust graph comparison. In *NIPS*, 2134–2142.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018a. Graph convolutional neural networks for web-scale recommender systems. *KDD*.

Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018b. Hierarchical graph representation learning with differentiable pooling. In *NIPS*.

Zhang, M.; Cui, Z.; Neumann, M.; and Yixin, C. 2018. An end-to-end deep learning architecture for graph classification. In *AAAI*, 4428–4435.