More Powerful Graph Neural Networks with Nesting

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph neural network (GNN)'s success in graph classification is closely related to the Weisfeiler-Lehman (1-WL) algorithm. By iteratively aggregating neighboring node features to a center node, both 1-WL and GNN obtain a node representation that encodes a *rooted subtree* around the center node. These rooted subtree representations are then pooled into a single representation to represent the whole graph. However, rooted subtrees are of limited expressiveness to represent a non-tree graph. To address it, we propose Nested Graph Neural Networks (NGNNs). NGNN represents a graph with *rooted subgraphs* instead of rooted subtrees, so that two graphs sharing many identical subgraphs (rather than subtrees) tend to have similar representations. The key is to make each node representation encode a subgraph around it more than a subtree. To achieve this, NGNN extracts a local subgraph around each node and applies a *base GNN* to each subgraph to learn a subgraph representation. The whole-graph representation is then obtained by pooling these subgraph representations. We provide a rigorous theoretical analysis showing that NGNN is strictly more powerful than 1-WL. In particular, we proved that NGNN can discriminate almost all r-regular graphs, where 1-WL always fails. Moreover, unlike other more powerful GNNs, NGNN only introduces a constant factor in time complexity compared to standard GNNs. NGNN is a plug-and-play framework that can be combined with various base GNNs. We test NGNN with different base GNNs on several benchmark datasets. NGNN uniformly improves their performance and shows highly competitive performance on all datasets.

1 Introduction

Graph is an important tool to model relational data in the real world. Representation learning over graphs has become a popular topic of machine learning in recent years. While network embedding methods, such as DeepWalk [1], can learn node representations well, they fail to generalize to whole-graph representations, which are crucial for applications such as graph classification, molecule modeling, and drug discovery. On the contrary, although traditional graph kernels [2–7] can be used for graph classification, they define graph similarity often in a heuristic way, which is not parameterized and lacks some flexibility to deal with features.

In this context, graph neural networks (GNNs) have regained people's attention and become the state-of-the-art graph representation learning tool [8–17]. GNNs use message passing to propagate features between connected nodes. By iteratively aggregating neighboring node features to the center node, GNNs learn node representations encoding their local structure and feature information. These node representations can be further pooled into a graph representation, enabling graph-level tasks such as graph classification. In this paper, we will use *message passing GNNs* to denote this class of GNNs based on repeated neighbor aggregation [18], in order to distinguish them from some high-order GNN variants [19–21] where the effective message passing happens between high-order node tuples instead of nodes.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

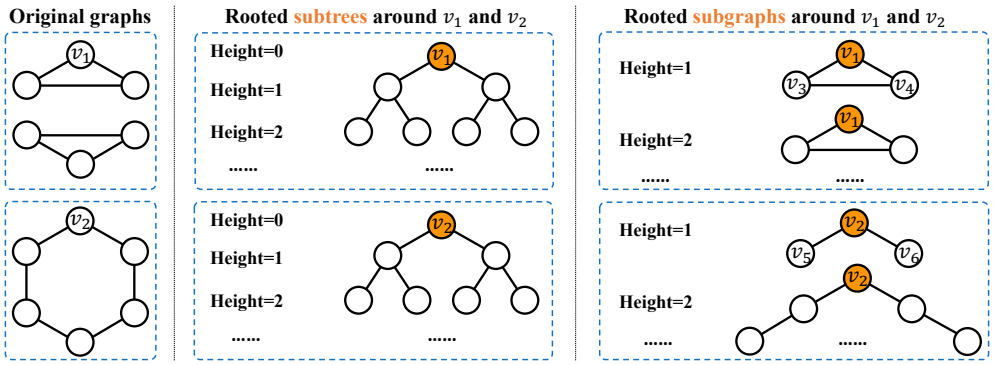

Figure 1: The top graph is disconnected, while the bottom graph is connected. Both 1-WL and message passing GNNs cannot differentiate them, since all nodes in the two graphs share identical rooted subtrees at any height. In comparison, we can discriminate the two graphs by comparing their height-1 rooted subgraphs around nodes.

GNNs' message passing scheme mimics the 1-dimensional Weisfeiler-Lehman (1-WL) algorithm [22], which iteratively refines a node's color according to its current color and the multiset of its neighbors' colors. This procedure essentially encodes a rooted subtree around each node into its final color, where the rooted subtree is constructed by recursively expanding the neighbors of the root node. One critical reason for GNN's success in graph classification is because, two graphs sharing many identical or similar rooted subtrees are more likely classified into the same class, which actually aligns with the inductive bias that two graphs are similar if they have many common substructures [23].

Despite this, rooted subtrees are still limited in terms of expressing **all possible substructures** that can appear in a graph. It is likely that two graphs, despite sharing a lot of identical rooted subtrees, are not similar at all because their other substructure patterns are not similar. Take the two graphs in Figure 1 as an example. If we apply 1-WL or a message passing GNN to them, the two graphs will always have the same representation no matter how many iterations/layers we use. This is because **all** nodes in the two graphs have identical rooted subtrees across **all** tree heights. However, the two graphs are quite different from a holistic perspective. The top graph is composed of two triangles, while the bottom graph is a connected circle. The intrinsic reason for such a failure is that rooted subtrees have limited expressiveness for representing general graphs, especially those with cycles.

To address this issue, we propose Nested Graph Neural Networks (NGNNs). The core idea is, instead of encoding a rooted subtree, we want the final representation of a node to better encode a **rooted subgraph** (local h-hop subgraph) around it. The subgraph is not restricted to be of any particular graph type such as tree, but serves as a general description of the local neighborhood around a node. Rooted subgraphs offer much better representation power than rooted subtrees, e.g., we can easily discriminate the two graphs in Figure 1 by only comparing their height-1 rooted subgraphs.

To represent a graph with rooted subgraphs, NGNN uses **two** levels of GNNs: a base (inner) GNN and an outer GNN. By extracting a local subgraph around each node, NGNN first applies the base GNN to each node's subgraph independently. Then, a subgraph pooling layer is applied to each subgraph to aggregate the intermediate node representations into a subgraph representation. This subgraph representation is used as the final representation of the root node. Rather than encoding a rooted subtree, this final node representation encodes the local subgraph around it, which contains more information than a subtree. Finally, all the final node representations are further fed into the outer GNN to learn a representation for the entire graph.

One may wonder that the base GNN seems to still learn only rooted subtrees if it is message-passing-based. Then why is NGNN more powerful than GNN? One key reason lies in the subgraph pooling layer. Take the height-1 rooted subgraphs around v_1 and v_2 in Figure 1 as an example. Suppose we use one message passing layer in the base GNN. Then both v_1 and v_2 will encode identical height-1 rooted subtrees (an open triplet), thus having the same intermediate representation. Nevertheless, nodes v_3 and v_4 in v_1's subgraph will encode different rooted subtrees from nodes v_5 and v_6 in v_2's subgraph. After applying a pooling layer (such as sum or mean pooling) over the intermediate node representations within the subgraphs, we can discriminate the rooted subgraphs around v_1 and v_2.

The NGNN framework has multiple exclusive advantages. Firstly, it allows freely choosing the base GNN, and can enhance the base GNN's representation power in a plug-and-play fashion.

Theoretically, we proved that NGNN is more powerful than message passing GNNs and 1-WL by being able to discriminate almost all r-regular graphs (where 1-WL always fails). Secondly, by extracting rooted subgraphs, NGNN allows augmenting the initial features of a node with subgraph-specific structural features, in contrast to standard GNNs which use the same initial features for a node no matter which root node's subgraph it is within. Thirdly, unlike other more powerful graph neural networks, especially those based on higher-order WL tests [19–21, 24], NGNN only incurs a constant time higher time complexity compared to standard message passing GNNs, thus still maintaining good scalability. We demonstrate the effectiveness of the NGNN framework in various synthetic/real-world graph classification/regression datasets. NGNN consistently enhances the base GNNs' performance, achieving highly competitive results on all datasets. In particular, NGNN achieves a new state-of-the-art result (Average Precision of 30.07) on the challenging `ogbg-molpcba`.

2 Preliminaries

2.1 Notation and problem definition

We consider the graph classification/regression problem. Given a graph $G = (V, E)$ where $V = \{1, 2, \dots n\}$ is the node set and $E \subseteq V \times V$ is the edge set, we aim to learn a function mapping G to its class or target variable y. The nodes and edges in G can have feature vectors associated with them, denoted by $\boldsymbol{x}_i$ (for node i) and $\boldsymbol{e}_{ij}$ (for edge (i, j)), respectively.

2.2 Weisfeiler-Lehman test

The Wesfeiler-Lehman (1-WL) test [22] is a popular algorithm for graph isomorphism checking. The classical 1-WL works as follows. At first, all nodes receive a color 1. Each node collects its neighbors' colors into a multiset. Then, 1-WL will update each node's color so that two nodes get the same new color if and only if their current colors are the same and they have identical multisets of neighbor colors. Repeat this process until the number of colors does not increase between two iterations. Then, 1-WL will return that two graphs are non-isomorphic if their node colors are different at some iteration, or fail to determine whether they are non-isomorphic. See [7, 25] for more detail.

1-WL essentially encodes the rooted subtrees around each node at different heights into its color representations. Figure 1 middle shows the rooted subtrees around v_1 and v_2. Two nodes will have the same color at iteration h if and only if their height-h rooted subtrees are the same.

3 Nested Graph Neural Network

In this section, we introduce our Nested Graph Neural Network (NGNN) framework and theoretically demonstrate its higher representation power than message passing GNNs.

3.1 Limitations of the message passing GNNs

Most existing GNNs follow the message passing framework [18]: given a graph G, each node's hidden state $\boldsymbol{h}_v^{t+1}$ is updated based on its previous state $\boldsymbol{h}_v^t$ and the messages $\boldsymbol{m}_v^{t+1}$ from its neighbors

$$\boldsymbol{h}_v^{t+1} = U_t(\boldsymbol{h}_v^t, \boldsymbol{m}_v^{t+1}), \quad \text{where} \quad \boldsymbol{m}_v^{t+1} = \sum_{u \in N(v|G)} M_t(\boldsymbol{h}_v^t, \boldsymbol{h}_u^t, \boldsymbol{e}_{vu}). \tag{1}$$

Here M_t, U_t are the message and update functions at time stamp t, $\boldsymbol{e}_{vu}$ is the feature of edge (v, u), and $N(v|G)$ is the set of v's neighbors in graph G. The initial hidden states $\boldsymbol{h}_v^0$ are given by the raw node features $\boldsymbol{x}_v$. After T time stamps (iterations), the final node representations $\boldsymbol{h}_v^T$ are summarized into a whole-graph representation with a readout (pooling) function R (e.g., mean or sum):

$$\boldsymbol{h}_G = R(\{\boldsymbol{h}_v^T | v \in G\}). \tag{2}$$

Such a message passing (or neighbor aggregation) scheme iteratively aggregates neighbor information into a center node's hidden state, making it encode a local rooted subtree around the node. The final node representations will contain both the local structure and feature information around nodes, enabling node-level tasks such as node classification. After a pooling layer, these node representations

can be further summarized into a graph representation, enabling graph-level tasks. When there is no edge feature and the node features are from a countable space, it is shown that message passing GNNs are at most as powerful as the 1-WL test for discriminating non-isomorphic graphs [26, 19].

For an h-layer message passing GNN, it will give two nodes the same final representation if they have identical **height-h rooted subtrees** (i.e., both the structures and the features on the corresponding nodes/edges are the same). If two graphs have a lot of identical (or similar) rooted subtrees, they will also have similar graph representations after pooling. This insight is crucial for the success of modern GNNs in graph classification, because it aligns with the inductive bias that two graphs are similar if they have many common substructures. Such insight has also been used in designing the WL subtree kernel [7], a state-of-the-art graph classification method before GNNs.

However, message passing GNNs have several limitations. Firstly, rooted subtree is only one specific substructure. It is not general enough to represent arbitrary subgraphs, especially those with cycles due to the natural restriction from tree structure. Secondly, using rooted subtree as the elementary substructure results in a discriminating power bounded by the 1-WL test. For example, all n-node r-regular graphs cannot be discriminated by message passing GNNs. Thirdly, the initial node features $\boldsymbol{x}_v$ are the same for a node v no matter which root node's message passing function it attends. This prevents us from using root-node-specific features to augment the raw node features. We need to break through such limitations in order to design more powerful GNNs.

3.2 The NGNN framework

To address the above limitations, we propose the Nested Graph Neural Network (NGNN) framework. NGNN no longer aims to encode a rooted subtree around each node. Instead, in NGNN, each node's final representation encodes the general local subgraph information around it more than a subtree, so that two graphs sharing a lot of identical or similar *rooted subgraphs* will have similar representations.

Definition 1. ***(Rooted subgraph)*** *Given a graph G and a node v, the height-h rooted subgraph G_v^h of v is the subgraph induced from G by the nodes within h hops of v (including h-hop nodes).*

To make a node's final representation encode a rooted subgraph, we need to compute a subgraph representation. To achieve this, we resort to another GNN, which we call the *base GNN* of NGNN. For example, the base GNN can be simply a message passing GNN, which performs message passing **within** the rooted subgraph to learn an intermediate representation for **every** node of the subgraph, and then uses a pooling layer to summarize a subgraph representation from the intermediate node representations. This subgraph representation is used as the final representation of the root node in the original graph. Take node w as an example. We first perform T rounds of message passing within node w's rooted subgraph G_w^h:

$$\boldsymbol{h}_{v,G_w^h}^{t+1} = U_t(\boldsymbol{h}_{v,G_w^h}^{t}, \boldsymbol{m}_{v,G_w^h}^{t+1}), \text{ where } \boldsymbol{m}_{v,G_w^h}^{t+1} = \sum_{u \in N(v|G_w^h)} M_t(\boldsymbol{h}_{v,G_w^h}^{t}, \boldsymbol{h}_{u,G_w^h}^{t}, \boldsymbol{e}_{vu}). \tag{3}$$

Here M_t, U_t are the message and update functions of the base GNN at time stamp t, $N(v|G_w^h)$ denotes the set of v's neighbors within w's rooted subgraph G_w^h, and $\boldsymbol{h}_{v,G_w^h}^{t+1}$ and $\boldsymbol{m}_{v,G_w^h}^{t+1}$ denote node v's hidden state and message **specific to** rooted subgraph G_w^h at time stamp $t+1$. Note that when node v attends different nodes' rooted subgraphs, its hidden states and messages will also be **different**. This is in contrast to standard GNNs where a node's hidden state and message at time t is the same regardless of which root node it contributes to. For example, $\boldsymbol{h}_v^{t+1}$ and $\boldsymbol{m}_v^{t+1}$ in Eq. 1 does not depend on any particular rooted subgraph.

After T rounds of message passing, we apply a *subgraph pooling layer* to summarize a subgraph representation $\boldsymbol{h}_{G_w^h}$ from the intermediate node representations $\{\boldsymbol{h}_{v,G_w^h}^{T} | v \in G_w^h\}$.

$$\boldsymbol{h}_w := \boldsymbol{h}_{G_w^h} = R_0(\{\boldsymbol{h}_{v,G_w^h}^{T} | v \in G_w^h\}), \tag{4}$$

where R_0 is the subgraph pooling layer. This subgraph representation $\boldsymbol{h}_{G_w^h}$ will be used as root node w's final representation $\boldsymbol{h}_w$ in the original graph. The base GNN is simultaneously and independently applied to all nodes' rooted subgraphs to return a node representation for all nodes in the original graph. With such node representations, the *outer GNN* further aggregates them into a graph representation of the whole graph, with another *graph pooling layer* R_1:

$$\boldsymbol{h}_G := R_1(\{\boldsymbol{h}_w | w \in G\}). \tag{5}$$

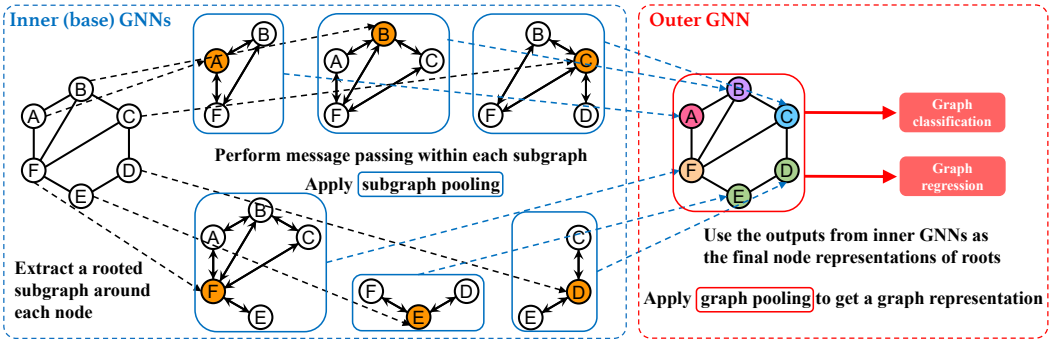

Figure 2: The NGNN framework. NGNN first extracts a rooted subgraph around each node. It then applies a base GNN with a subgraph pooling layer to each rooted subgraph independently. The subgraph representation is used as the root node's final representation in the original graph. Then, a graph pooling layer is used to summarize the final node representations into a graph representation.

The Nested GNN framework can be understood as a two-level GNN, or a **GNN of GNNs**—the inner subgraph-level GNNs (base GNNs) are used to learn node representations from their rooted subgraphs, while the outer graph-level GNN is used to return a whole-graph representation from the inner GNNs' outputs. The inner GNNs all share the same parameters which are trained end-to-end with the outer GNN. Figure 2 depicts the overall NGNN framework.

Compared to message passing GNNs, NGNN changes the "receptive field" of each node from a rooted subtree to a rooted subgraph, in order to capture better local substructure information. The rooted subgraph is read by a base GNN to learn a subgraph representation. Finally, the outer GNN reads the subgraph representations output by the base GNNs to return a graph representation.

Note that, when we apply the base GNN to a rooted subgraph, this rooted subgraph is extracted (copied) out of the original graph and treated as a completely independent graph from the other rooted subgraphs and the original graph. This allows the same node to have **different** representations within different rooted subgraphs. For example, in Figure 2, the same node B appears in four different rooted subgraphs. Sometimes it is the root node, while other times it is a 1-hop neighbor of the root node. NGNN enables learning different representations for the same node when it appears in different rooted subgraphs, in contrast to standard GNNs where a node only has one single representation at one time stamp (Eq. 1). Similarly, NGNN also enables using different initial features for the same node when it appears in different rooted subgraphs. This allows us to customize a node's initial features based on its structural role within a rooted subgraph, as opposed to using the same initial features for a node across all rooted subgraphs. For example, we can augment node B's initial features with the distance between node B and the root—when node B is the root node, we give it an additional feature 0; and when B is a k-hop neighbor of the root, we give it an additional feature k. Such feature augmentation helps better capture a node's structural role within a rooted subgraph. It is an exclusive advantage of NGNN and is **not** possible in standard GNNs.

3.3 The representation power of NGNN

We want to theoretically characterize the additional power of NGNN as opposed to message passing GNNs. We focus on the power to distinguish different graph structures. As their representation power is limited by 1-WL, message passing GNNs fail to distinguish all pairs of n-sized r-regular graphs, unless discriminative node features can be leveraged. In contrast, we prove that NGNN can distinguish almost all pairs of n-sized r-regular graphs regardless of node features.

Definition 2. *If the message passing (Eq. 3) and the two-level graph pooling (Eqs. 4,5) are all injective given input from a countable space, then the NGNN is called* proper.

A proper NGNN always exists due to the representation power of fully-connected neural networks used for message passing and Deep Set for graph pooling [27]. For all pairs of graphs that 1-WL can discriminate, there always exists a proper NGNN that can also discriminate them, because two graphs discriminated by 1-WL means they must have different multisets of rooted subtrees at some height h, while a rooted subtree is always included in a rooted subgraph with the same height.

Now we present our main theorem.

Theorem 1. *Consider all pairs of n-sized r-regular graphs, where $3 \leq r < (2\log n)^{1/2}$. For any small constant $\epsilon > 0$, there exists a proper NGNN using at most $\lceil(\frac{1}{2}+\epsilon)\frac{\log n}{\log(r-1-\epsilon)}\rceil$-height rooted subgraphs and $\lceil\epsilon\frac{\log n}{\log(r-1-\epsilon)}\rceil$-layer message passing, which distinguishes almost all $(1-o(1))$ such pairs of graphs.*

We include the proof in Appendix A. Theorem 1 has three implications. Firstly, since NGNN can discriminate almost all r-regular graphs where 1-WL always fails, it is **strictly more powerful** than 1-WL and message passing GNNs. Secondly, it implies that NGNN does not need to extract subgraphs with a too large height (about $\frac{1}{2}\frac{\log n}{\log(r-1)}$) to be more powerful. Moreover, NGNN is already powerful with very few layers, i.e., an arbitrarily small constant times $\frac{\log n}{\log(r-1)}$ (as few as 1 layer). This benefit comes from the subgraph pooling (Eq. 4), freeing us from using deep base GNNs. We further conduct a simulation experiment in Appendix C to verify Theorem 1 by testing how well NGNN discriminates r-regular graphs in practice. The results match almost perfectly with our theory.

Although NGNN is strictly more powerful than 1-WL and 2-WL (1-WL and 2-WL have the same discriminating power [20]), it is unclear whether NGNN is more powerful than 3-WL. Our initial analysis shows both NGNN and 3-WL cannot discriminate strongly-regular graphs with the same parameters [28]. We leave the exact comparison between NGNN and 3-WL to future work.

3.4 Discussion

Base GNN. NGNN is a general framework to increase the representation power of GNNs. For the base GNN, we are not restricted to message passing GNNs as described in Section 3.2. For example, we can also use GNNs matching the power of higher-dimensional WL tests, such as 1-2-3-GNN [19] and PPGN/Ring-GNN [20, 21], as the base GNN. In fact, one limitation of these high-order GNNs is their $\mathcal{O}(n^3)$ complexity. Using the NGNN framework we can greatly alleviate this. Suppose a rooted subgraph has at most c nodes, then by applying a high-order GNN to all n rooted subgraphs, we can reduce the time complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(nc^3)$.

Complexity. We compare the time complexity of NGNN (using a message passing GNN as the base GNN) with a standard message passing GNN. Suppose the graph has n nodes with a maximum degree d, and the maximum number of nodes in a rooted subgraph is c. Each message passing iteration in a standard message passing GNN takes $\mathcal{O}(n \cdot d)$ operations. In NGNN, we need to perform message passing over all n nodes' rooted subgraphs, which takes $\mathcal{O}(nc \cdot d)$. We will keep c small so that the base GNN focuses on learning local subgraph patterns.

4 Related work

Understanding GNN's representation power is a fundamental problem in GNN research. Xu et al. [26] and Morris et al. [19] first proved that the discriminating power of message passing GNNs is bounded by the 1-WL test, namely they cannot discriminate two non-isomorphic graphs that 1-WL fails to discriminate (such as r-regular graphs). Since then, there is increasing effort in enhancing GNN's discriminating power beyond 1-WL [19, 21, 20, 29–33, 24]. Many GNNs have been proposed to mimic higher-dimensional WL tests, such as 1-2-3-GNN [19], Ring-GNN [21] and PPGN [20]. However, these models generally require learning the representations of all node subsets of certain cardinality (e.g., node pairs, node triples and so on), thus cannot leverage the sparsity of graph structure and are difficult to scale to large graphs. Some works study the universality of GNNs for approximating any invariant or equivariant functions over graphs [34, 21, 35–37]. However, reaching universality would require polynomial(n)-order tensors, which hold more theoretical value than practical applicability. Relational Pooling (RP) [29] uses the ensemble of permutation-aware functions over graphs to reach universality, which requires exhausting all $n!$ permutations to achieve its theoretical power. Similarly, Dasoulas et al. [38] propose to augment nodes of identical attributes with different colors, which also requires exhausting all the coloring choices to reach universality.

Because of the high cost of mimicking high-dimensional WL tests, several works have been proposed to increase GNN's representation power within the message passing framework. Noticing that different neighbors are indistinguishable during neighbor aggregation, some works propose to add one-hot node index features or random features to GNNs [39, 40]. These methods work well when nodes naturally have distinct identities irrespective of the graph structure. However, although making

GNNs more discriminative, they also lose some of GNNs' generalization ability by not being able to guarantee nodes with identical neighborhoods to have the same embedding; the resulting models are also no longer permutation invariant. Repeating random initialization helps with avoiding such an issue but gets much slower convergence [41]. A notable exception is structural message-passing (SMP) [42], which propagates one-hot node index features to learn a global $n \times d$ feature matrix for each node. The feature matrix is further pooled to learn a permutation-invariant node representation.

On the contrary, some works propose to use structural features to augment GNNs without hurting the generalization ability of GNNs. SEAL [43, 44] and DE [30] use distance-based features, where a distance vector w.r.t. the target node set to predict is calculated for each node as their additional features. Our NGNN framework is naturally compatible with such distance-based features due to its independent rooted subgraph processing. GSN [31] uses the count of certain substructures to augment node/edge features, which also surpasses 1-WL theoretically. However, GSN needs a properly defined substructure set to incorporate domain-specific inductive biases, while NGNN aims to learn arbitrary substructures around nodes without the need to predefine a substructure set.

Concurrent to our work, You et al. [32] propose Identity-aware GNN (ID-GNN). ID-GNN uses different weight parameters between the center node and other context nodes during message passing, and is also beyond 1-WL. ID-GNN can be viewed as a special case of NGNN with 1) number of GNN layers equivalent to the height of the subgraph, 2) directly using the root representation without subgraph pooling, and 3) augmenting initial node features with 0/1 "identity". However, the power of ID-GNN only comes from the "identity" feature, while the power of NGNN comes from the subgraph pooling—without using any node features, NGNN is still provably more discriminative than 1-WL. Another similar work to ours is natural graph network (NGN) [45]. NGN argues graph convolution weights need not be shared among all nodes but only (locally) isomorphic nodes. If we view our distance-based node features as refining the graph convolution weights so that nodes within a center node's neighborhood are no longer treated symmetrically, then our NGNN reduces to an NGN.

The idea of independently performing message passing within k-hop neighborhood is also explored in k-hop GNN [46] and MixHop [47]. However, MixHop directly concatenates the aggregation results of neighbors at different hops as the root representation, which ignores the connections between other nodes in the rooted subgraph. k-hop GNN sequentially performs message passing for k-hop, $k-1$-hop, ..., and 0-hop node (the update of $(i-1)$-hop nodes depend on the updated states of i-hop nodes), while NGNN simultaneously performs message passing for all nodes in the subgraph thus is more parallelizable. Both MixHop and k-hop GNN directly use the root node's representation as its final node representation. In contrast, NGNN uses a subgraph pooling to summarize all node representations within the subgraph as the final root representation, which distinguishes NGNN from other k-hop models. As Theorem 1 shows, the subgraph pooling enables using a much smaller number of message passing layers l (as small as 1) than the depth k of the subgraph, while MixHop and k-hop GNN always require $l \geq k$. MixHop and k-hop GNN also do not have the strong theoretical power of NGNN to discriminate r-regular graphs.

5 Experiments

In this section, we study the effectiveness of the NGNN framework for graph classification and regression tasks. In particular, we want to answer the following questions:
Q1 Is a practical NGNN able to reach its theoretical power for discriminating r-regular graphs?
Q2 How often does NGNN improve the performance of base GNNs?
Q3 How much improvement does NGNN bring to base GNNs than directly applying the base GNNs?
Q4 How does NGNN perform in comparison to state-of-the-art GNN methods in open benchmarks?
Q5 How much extra computation time does NGNN incur?
We answer **Q1** using a simulation experiment in Appendix C, and answer the other questions below.

5.1 Datasets

To answer **Q2** and **Q3**, we use the QM9 dataset [48, 49] and the TU datasets [50]. QM9 contains 130K small molecules. The task here is to perform regression on twelve targets representing energetic, electronic, geometric, and thermodynamic properties, based on the graph structure and node/edge features. TU contains five graph classification datasets including D&D [51], MUTAG [52], PROTEINS [51], PTC_MR [53], and ENZYMES [54]. We used the datasets provided by PyTorch

Table 1: Statistics and evaluation metrics of the QM9 and OGB datasets.

Dataset	#Graphs	Avg. #nodes	Avg. #edges	Split ratio	#Tasks	Task type	Metric
`QM9`	129,433	18.0	18.6	80/10/10	12	Regression	MAE
`ogbl-molhiv`	41,127	25.5	27.5	80/10/10	1	Classification	ROC-AUC
`ogbl-molpcba`	437,929	26.0	28.1	80/10/10	128	Classification	AP

Geometric [55], where for QM9 we performed unit conversions to match the units used by [19]. The evaluation metric is Mean Absolute Error (MAE) for QM9 and Accuracy (%) for TU.

To answer **Q4**, we use two Open Graph Benchmark (OGB) datasets [56], `ogbg-molhiv` and `ogbg-molpcba`.`ogbg-molhiv` contains 41K small molecules, the task of which is to classify whether a molecule inhibits HIV virus or not. ROC-AUC is used for evaluation. `ogbg-molpcba` contains 438K molecules with 128 classification tasks. The evaluation metric is Average Precision (AP) averaged over all the tasks. We include the statistics for QM9 and OGB datasets in Table 1.

5.2 Models

QM9. We use 1-GNN, 1-2-GNN, 1-3-GNN, and 1-2-3-GNN from [19] as both the baselines and the base GNNs of NGNN. Among them, 1-GNN is a standard message passing GNN with 1-WL power. 1-2-GNN is a GNN mimicking 2-WL, where message passing happens among 2-tuples of nodes. 1-3-GNN and 1-2-3-GNN mimic 3-WL, where message passing happens among 3-tuples of nodes. 1-2-GNN and 1-3-GNN use features computed by 1-GNN as initial node features, and 1-2-3-GNN uses the concatenated features from 1-2-GNN and 1-3-GNN. We additionally include numbers provided by [49]. Note that we omit more recent baselines [57–59] using advanced physical representations from angles, atom coordinates, and quantum mechanics, which may obscure the comparison of GNNs' pure graph regression performance. For NGNN, we uniformly use height-3 rooted subgraphs. For a fair comparison, the base GNNs in NGNN use exactly the same hyperparameters as when they are used alone, except for 1-GNN where we increase the number of message passing layers from 3 to 5 to make the number of layers larger than the subgraph height. For subgraph pooling and graph pooling layers, we uniformly use mean pooling. All other settings follow [19].

TU. We use four widely adopted GNNs as the baselines and the base GNNs of NGNN: GCN [12], GraphSAGE [60], GIN [26], and GAT [15]. Since TU datasets suffer from inconsistent evaluation standards [61], we uniformly use 4 message passing layers with 32 hidden dimensions each for all models, and train them for 100 epochs with a batch size of 128. We report the test set (10%) accuracy at the epoch with the smallest validation set (10%) loss. And the results are averaged over 10 runs. For NGNN, we uniformly use height-3 rooted subgraphs with mean pooling as the subgraph/graph pooling layers. All other hyperparameters are the same as when training the original base GNNs.

OGB. We use GNNs achieving top places on the OGB graph classification leaderboard (`https://ogb.stanford.edu/docs/leader_graphprop/`) as the baselines, including GCN [12], GIN [26], DeeperGCN [62], HIMP [63], PNA [64], DGN [33], GINE [65], and PHC-GNN [66]. Note that those high-order GNNs [19–21, 24] are not included here, because despite being theoretically more discriminative, these GNNs are **not** among the GNNs with the best empirical performance on modern large-scale graph benchmarks, and their $\mathcal{O}(n^3)$ complexity also raises a scalability issue. For NGNN, we use GIN as the base GNN (although GIN is not among the strongest baselines here). Some baselines additionally use the virtual node technique [18, 11, 67], which are marked by "*". For NGNN, we search the subgraph height h in $\{3, 4, 5\}$, and the number of layers in $\{4, 5, 6\}$. We train the NGNN models for 100 and 150 epochs for `ogbg-molhiv` and `ogbg-molpcba`, respectively, and report the validation and test scores at the best validation epoch. We also find that our models are subject to high performance variance across epochs, likely due to the increased expressiveness. Thus, we save a model checkpoint every 10 epochs, and additionally report the ensemble performance by averaging the predictions from all checkpoints. The final hyperparameter choices and more details about the experimental settings are included in Appendix D. All results are averaged over 10 runs.

For all NGNN models, we augment the initial features of a node with Distance Encoding (DE) [30], which uses the (generalized) distance between a node and the root as its additional feature, due to DE's successful applications in link-level tasks [43, 68]. Note that such feature augmentation is not applicable to the baseline models as discussed in Section 3.2. An ablation study on their effects are included in Appendix E.

Table 2: MAE results on QM9 (smaller the better). A colored cell means NGNN is better than the base GNN.

Target	Method										
	DTNN	**MPNN**	**1-GNN**	**1-2-GNN**	**1-3-GNN**	**1-2-3-GNN**	**Nested 1-GNN**	**Nested 1-2-GNN**	**Nested 1-3-GNN**	**Nested 1-2-3-GNN**	**Max. reduction**
μ	**0.244**	0.358	0.493	0.493	0.473	0.476	0.428	0.437	0.436	0.433	1.2×
α	0.95	0.89	0.78	0.27	0.46	0.27	0.29	0.278	**0.261**	0.265	2.7×
$\varepsilon_{\text{HOMO}}$	0.00388	0.00541	0.00321	0.00331	0.00328	0.00337	**0.00265**	0.00275	**0.00265**	0.00279	1.2×
$\varepsilon_{\text{LUMO}}$	0.00512	0.00623	0.00355	0.00350	0.00354	0.00351	0.00297	0.00271	**0.00269**	0.00276	1.3×
$\Delta\varepsilon$	0.0112	0.0066	0.0049	0.0047	0.0046	0.0048	**0.0038**	0.0039	0.0039	0.0039	1.8×
$\langle R^2 \rangle$	**17.0**	28.5	34.1	21.5	25.8	22.9	20.5	20.4	20.2	20.1	1.7×
ZPVE	0.00172	0.00216	0.00124	0.00018	0.00064	0.00019	0.00020	0.00017	0.00017	**0.00015**	6.2×
U_0	2.43	2.05	2.32	**0.0357**	0.6855	0.0427	0.295	0.252	0.291	0.205	7.9×
U	2.43	2.00	2.08	**0.107**	0.686	0.111	0.361	0.265	0.278	0.200	5.8×
H	2.43	2.02	2.23	0.070	0.794	**0.0419**	0.305	0.241	0.267	0.249	7.3×
G	2.43	2.02	1.94	0.140	0.587	**0.0469**	0.489	0.272	0.287	0.253	4.0×
C_v	0.27	0.42	0.27	0.0989	0.158	0.0944	0.174	0.0891	0.0879	**0.0811**	1.8×

Table 3: Accuracy results (%) on TU datasets.

	D&D	MUTAG	PROTEINS	PTC_MR	ENZYMES
#Graphs	1178	188	1113	344	600
Avg. #nodes	284.32	17.93	39.06	14.29	32.63
GCN	72.7±3.6	73.4±8.8	71.9±4.4	57.8±5.0	27.7±7.2
GraphSAGE	72.2±4.0	74.0±11.5	72.0±3.8	55.8±6.2	28.5±7.2
GIN	70.0±3.8	82.0±11.4	72.2±6.1	56.7±5.8	**36.0**±3.5
GAT	70.3±3.3	72.9±10.5	71.7±4.1	59.0±5.5	28.3±6.8
Nested GCN	75.4±3.1	75.9±11.6	**73.9**±4.2	57.8±5.7	30.7±4.2
Nested GraphSAGE	**76.2**±4.3	72.8±10.0	73.0±3.1	**61.1**±3.9	29.2±5.5
Nested GIN	75.5±5.2	**85.2**±8.1	72.1±3.2	56.2±7.3	33.2±6.5
Nested GAT	73.6±4.5	77.6±10.4	72.9±4.0	58.4±6.1	29.8±5.7
Max. improvement	5.5%	4.7%	2.0%	5.3%	3.0%

Table 4: Results on OGB datasets (* virtual node).

	`ogbg-molhiv` ROC-AUC (%)		`ogbg-molpcba` AP (%)	
Method	Validation	**Test**	Validation	**Test**
CCN*	83.84±0.91	75.99±1.19	24.95±0.42	24.24±0.34
GIN*	84.79±0.68	77.07±1.49	27.98±0.25	27.03±0.23
DeeperGCN*	–	–	29.20±0.25	27.81±0.38
HIMP	–	78.80±0.82	–	–
PNA	85.19±0.99	79.05±1.32	–	–
DGN	84.70±0.47	79.70±0.97 –	–	
GINE*	–	–	30.65±0.30	29.17±0.15
PHC-GNN	82.17±0.89	79.34±1.16	30.68±0.25	29.47±0.26
Nested GIN*	83.17±1.99	78.34±1.86	29.15±0.35	28.32±0.41
Nested GIN* (ens)	80.80±2.78	**79.86**±1.05	30.59±0.56	**30.07**±0.37

5.3 Results and discussion

We show the experimental results on QM9 in Table 2. If the Nested version of a GNN achieves a better result than its basic version, we will color that cell with light green. As we can see, NGNN brings performance gains to all base GNNs on most targets, sometimes by large margins. We also show the results on TU in Table 3. NGNNs also show improvement over their base GNNs in most cases. These results answer **Q2**, indicating that NGNN is a general framework for improving a GNN's power. We further compute the maximum reduction of MAE for QM9 and maximum absolute improvement of accuracy for TU before and after applying NGNN. NGNN reduces the MAE by up to 7.9 times for QM9, and increases the accuracy by up to 5.5% for TU. These results answer **Q3**, indicating that NGNN can bring significant improvement to base GNNs.

To answer **Q4**, we compare Nested GIN with leading methods on the OGB leaderboard. The results are shown in Table 4. Nested GIN achieves highly competitive performance with these leading GNN models, albeit using a relatively weak base GNN (GIN). Compared to GIN alone, Nested GIN shows clear performance gains. The ensemble Nested GIN achieves test scores of 79.86 and 30.07 on `ogbg-molhiv` and `ogbg-molpcba`, respectively, which outperform all the baselines. In particular, for the challenging `ogbg-molpcba`, this is the first time that a method can achieve over 30.00 test AP averaged over 128 tasks, which ranks the 1st on the leaderboard at the time of submission. These significant results demonstrate the great empirical performance of NGNN, even compared to heavily tuned open leaderboard models. We believe NGNN could be even better with a stronger base GNN.

To answer **Q5**, we report the training time per epoch for GIN and Nested GIN on OGB datasets. On `ogbg-molhiv`, GIN takes 54s per epoch, while Nested GIN takes 183s per epoch. On `ogbg-molpcba`, GIN takes 10min per epoch, while Nested GIN takes 20min. This verifies NGNN's constant-factor higher time complexity. The additional complexity comes from independently learning better node representations from rooted subgraphs, which is a trade-off for the higher expressivity. Finally, we point out one limitation of NGNN. Currently, NGNN does not scale to graph datasets with an average node number over 400 (such as REDDIT-BINARY) due to copying a rooted subgraph for each node to the GPU memory. Reducing batch size or subgraph height helps, but also leads to performance degradation. We leave the exploration of memory-efficient NGNN to the future work.

6 Conclusions

We have proposed Nested Graph Neural Network (NGNN), a general framework for improving GNN's representation power. NGNN learns node representations encoding rooted subgraphs more than rooted subtrees. Theoretically, we prove NGNN can discriminate almost all r-regular graphs which 1-WL always fails to do. Empirically NGNN consistently improves the performance of various base GNNs across different datasets while only incurring a constant-factor higher time complexity.

References

[1] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[2] David Haussler. Convolution kernels on discrete structures. Technical report, Citeseer, 1999.

[3] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, volume 5, pages 488–495, 2009.

[4] Risi Kondor, Nino Shervashidze, and Karsten M Borgwardt. The graphlet spectrum. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 529–536. ACM, 2009.

[5] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *5th IEEE International Conference on Data Mining*, pages 8–pp. IEEE, 2005.

[6] Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2): 209–245, 2016.

[7] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep): 2539–2561, 2011.

[8] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[9] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[10] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[11] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[12] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[13] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3837–3845, 2016.

[14] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2702–2711, 2016.

[15] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[16] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, pages 4438–4445, 2018.

[17] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

[18] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[19] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[20] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pages 2156–2167, 2019.

[21] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems*, pages 15894–15902, 2019.

[22] Boris Weisfeiler and AA Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

[23] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

[24] Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. 2020.

[25] Muhan Zhang and Yixin Chen. Weisfeiler-lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 575–583. ACM, 2017.

[26] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[27] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

[28] Andries E Brouwer and Willem H Haemers. Strongly regular graphs. In *Spectra of Graphs*, pages 115–149. Springer, 2012.

[29] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673. PMLR, 2019.

[30] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding–design provably more powerful gnns for structural representation learning. *arXiv preprint arXiv:2009.00142*, 2020.

[31] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*, 2020.

[32] Jiaxuan You, Jonathan Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. *arXiv preprint arXiv:2101.10320*, 2021.

[33] Dominique Beaini, Saro Passaro, Vincent Létourneau, William L Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. *arXiv preprint arXiv:2010.02863*, 2020.

[34] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

[35] Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *International conference on machine learning*, pages 4363–4371. PMLR, 2019.

[36] Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. *arXiv preprint arXiv:1905.04943*, 2019.

[37] Waïss Azizian and Marc Lelarge. Characterizing the expressive power of invariant and equivariant graph neural networks. *arXiv preprint arXiv:2006.15646*, 2020.

[38] George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. Coloring graph neural networks for node disambiguation. *arXiv preprint arXiv:1912.06058*, 2019.

[39] Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019.

[40] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. *arXiv preprint arXiv:2002.03155*, 2020.

[41] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

[42] Clément Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing. *arXiv e-prints*, pages arXiv–2006, 2020.

[43] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.

[44] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Revisiting graph neural networks for link prediction. *arXiv preprint arXiv:2010.16103*, 2020.

[45] Pim de Haan, Taco Cohen, and Max Welling. Natural graph networks. *arXiv preprint arXiv:2007.08349*, 2020.

[46] Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. k-hop graph neural networks. *Neural Networks*, 130:195–205, 2020.

[47] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019.

[48] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

[49] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

[50] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. URL `http://graphkernels.cs.tu-dortmund.de`.

[51] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

[52] Asim Kumar Debnath, de Compadre RL Lopez, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

[53] Hannu Toivonen, Ashwin Srinivasan, Ross D King, Stefan Kramer, and Christoph Helma. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics*, 19(10):1183–1193, 2003.

[54] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004.

[55] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[56] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[57] Brandon Anderson, Truong-Son Hy, and Risi Kondor. Cormorant: Covariant molecular neural networks. *arXiv preprint arXiv:1906.04015*, 2019.

[58] Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. *arXiv preprint arXiv:2003.03123*, 2020.

[59] Zhuoran Qiao, Matthew Welborn, Animashree Anandkumar, Frederick R Manby, and Thomas F Miller III. Orbnet: Deep learning for quantum chemistry using symmetry-adapted atomic-orbital features. *The Journal of Chemical Physics*, 153(12):124111, 2020.

[60] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.

[61] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

[62] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

[63] Matthias Fey, Jan-Gin Yuen, and Frank Weichert. Hierarchical inter-message passing for learning on molecular graphs. *arXiv preprint arXiv:2006.12179*, 2020.

[64] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

[65] Rémy Brossard, Oriel Frigo, and David Dehaene. Graph convolutions that can finally model local structure. *arXiv preprint arXiv:2011.15069*, 2020.

[66] Tuan Le, Marco Bertolini, Frank Noé, and Djork-Arné Clevert. Parameterized hypercomplex graph neural networks for graph classification. *arXiv preprint arXiv:2103.16584*, 2021.

[67] Katsuhiko Ishiguro, Shin-ichi Maeda, and Masanori Koyama. Graph warp module: an auxiliary module for boosting the power of graph neural networks. *arXiv preprint arXiv:1902.01020*, 2019.

[68] Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=ByxxgCEYDS`.

[69] Hongyang Gao and Shuiwang Ji. Graph u-nets. *arXiv preprint arXiv:1905.05178*, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A.
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We will release the code after cleaning it up.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix D.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [No] The license can be found in their github.

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The packages we used are all open-sourced.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] There are no personal information in the datasets.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]