Korea Advanced Institute of Technology

Department of Mathematical Sciences

Contemplation on *How powerful are graph neural networks?*

Author:
Junghyun Lee

Instructor:
Prof. Chang Dong Yoo

Final Project - Graph

EE531: Statistical Learning Theory, Fall 2019

December 25, 2019

Abstract

This writing is an extensive outline of the paper *How powerful are graph neural networks?* by Xu *et al.*, including summaries of the theorems/lemmas and additional explanation on some of the literatures/points that the paper omitted.

(This is to be accompanied by the pdf file used in the final presentation. Also, the theorem/lemma/corollary numbering is arbitrary, but the statements are exact.)

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

GNN, short for Graph Neural Network, has revolutionized the field of representation learning for graph-structured datas. Scarselli *et al.*[9] developed the (probably) earliest GNN model based on information diffusion mechanism. Since then, numerous variants of GNN models have come out, and the reader is referred to a recent survey of GNNs by Wu *et al.*[15].

Even though recent advances in GNNs have achieved state-of-the-art performance in many tasks such as node classification, link prediction, graph classification...etc., their designing have mostly been based upon empirical intuition, heuristics, and experimental trial-and-error. Therefore, there isn't much theoretical understanding of the properties and limitations of GNNs, and formal analysis of GNNs' representational capacity is limited.[16] Precisely, this work presents a theoretical framework for analyzing the representational power of GNNs i.e. how expressive different GNN variants are in learning to represent and distinguish between different graph structures.

1.2 Previous / Related Works

There hasn't been much work regarding this topic of theoretical interest. In his other work, Scarselli et al.[10] showed that his GNN model[9] can approximate measurable functions in probability. (Theorem 2-4 in his paper)

Lei et al.[8] showed that their architecture lies in the RKHS(reproducing kernel Hilbert space) of graph kernels, but do not study explicitly which graph it can distinguish.

These works focus on a specific architecture and **do not easily generalize to other architectures**.

Chapter 2

Preliminaries

2.1 GNNs

Consider a graph, $G = (V, E)$, or a data with graph structure. Then each node $v \in V$ is associated with a *node feature vector*, denoted as X_v. There are two tasks of interest where GNN is commonly used:

Problem (Node Classification Problem)
Each $v \in V$ has an associated label y_v.
Goal: *Learn a representation vector h_v of v such that $y_v = f(h_v)$ i.e. such that v's label can be predicted.*

Problem (Graph Classification Problem)
A set of graphs $\{G_1, \ldots, G_N\} \subset \mathcal{G}$ is given, along with their labels $\{y_1, \ldots, y_N\} \subset \mathcal{Y}$.
Goal: *Learn a representation vector h_G of G such that $y_G = f(h_G)$ i.e. such that G's label can be predicted.*

Modern GNNs follow a **neighborhood aggregation strategy** (message passing strategy) in which it iteratively updates the representation of a node by aggregating representations of its neighbors.

Denote $h_v^{(k)}$ as the feature vector of node v at the k-th iteration/layer, and let us initialize it as $h_v^{(0)} = X_v$. Then in this framework, the k-th layer of a GNN can be written as

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}_G(v)\right\}\right)$$
$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

Different choices of AGGREGATE$^{(k)}$ and COMBINE$^{(k)}$ have led to different GNN variants/architectures.

Architecture (GraphSAGE[4])

$$a_v^{(k)} = \text{MAX}\left(\left\{\text{ReLU}\left(W h_u^{(k-1)}\right) : u \in \mathcal{N}_G(v)\right\}\right)$$
$$h_v^{(k)} = W\left[h_v^{(k-1)}, a_v^{(k)}\right]$$

Architecture (Graph Convolutional Networks(GCN)[7])

$$h_v^{(k)} = \text{ReLU}\left(W\,\text{MEAN}\left\{h_u^{(k-1)} : u \in \mathcal{N}_G(v) \cup \{v\}\right\}\right)$$

Observe that for node classification tasks, the final node representation $h_v^{(K)}$ is directly used for prediction. However for graph classification tasks, this is not the case i.e. some additional work has to be done to "process" the final node representation to obtain the entire graph's representation. This is done by aggregating the final node representations by READOUT function:

$$h_G = \text{READOUT}\left(\left\{h_v^{(K)} : v \in V\right\}\right)$$

(READOUT can be a simple permutation invariant function, or something more sophisticated[19][21])

2.2 WL test

Now consider the following problem:

Problem (Graph Isomorphism (GI))
Input: *Two finite graphs G_1 and G_2*
Question: $G_1 \cong G_2$*?*

This ubiquitous problem appears everywhere: from mathematical logic, theory of computation, machine learning, to seemingly unrelated fields such as computer vision. And this seemingly harmless problem has harassed researchers for decades!

It is currently known that GI can be solved in quasipolynomial time *i.e.* in $O\left(2^{O((\log n)^c)}\right)$ $(c > 0)$ time[1]. The precise statement is as follows:

Theorem 2.2.1 (Babai, 2015)
The Graph Isomorphism problem ... can be solved in quasipolynomial time.

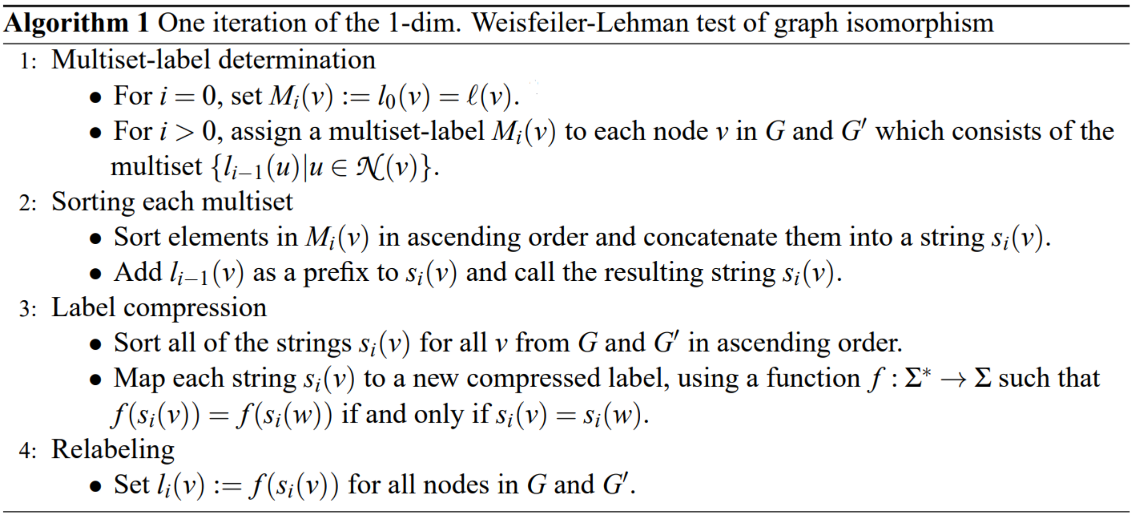
But this is not practical. In practice, other less efficient algorithms are used, such as algorithms by McKay (1981), Schmidt & Druffel (1976), Ullman (1976)...etc.

Here, we shall talk about one specific combinatorial algorithm for GI: the Weisfeiler-Lehman test of graph isomorphism[14], or simply WL test.

WL test is proved to be successful (and computationally efficient) in isomorphism testing for a broad class of graphs[2] There are some (corner) cases (ex. regular graphs) when the WL test fails[3].

The reason the people working on GNNs got so interested in the WL test is that its 1-dimensional form (a.k.a. "naïve vertex refinement") is *based on neighbor aggregations*, analogous to the GNNs! (And that is why we'll be focused only on the 1-dim case)

Let (G, l) be a labeled graph i.e. a graph G with an endowed node coloring $l : V(G) \to \Sigma$. (Σ: arbitrary codomain) Here is the full algorithm for the 1-dim WL test[11]:



Algorithm 1 One iteration of the 1-dim. Weisfeiler-Lehman test of graph isomorphism

```
1: Multiset-label determination
     • For $i = 0$, set $M_i(v) := l_0(v) = \ell(v)$.
     • For $i > 0$, assign a multiset-label $M_i(v)$ to each node $v$ in $G$ and $G'$ which consists of the
       multiset $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$.
2: Sorting each multiset
     • Sort elements in $M_i(v)$ in ascending order and concatenate them into a string $s_i(v)$.
     • Add $l_{i-1}(v)$ as a prefix to $s_i(v)$ and call the resulting string $s_i(v)$.
3: Label compression
     • Sort all of the strings $s_i(v)$ for all $v$ from $G$ and $G'$ in ascending order.
     • Map each string $s_i(v)$ to a new compressed label, using a function $f : \Sigma^* \to \Sigma$ such that
       $f(s_i(v)) = f(s_i(w))$ if and only if $s_i(v) = s_i(w)$.
4: Relabeling
     • Set $l_i(v) := f(s_i(v))$ for all nodes in $G$ and $G'$.
```

Figure 2.1: One iteration of 1-dim WL test

(This why this 1-dim version is commonly called the *color refinement algorithm*)

To relate this to GNN, we need one more concept; graph kernel.

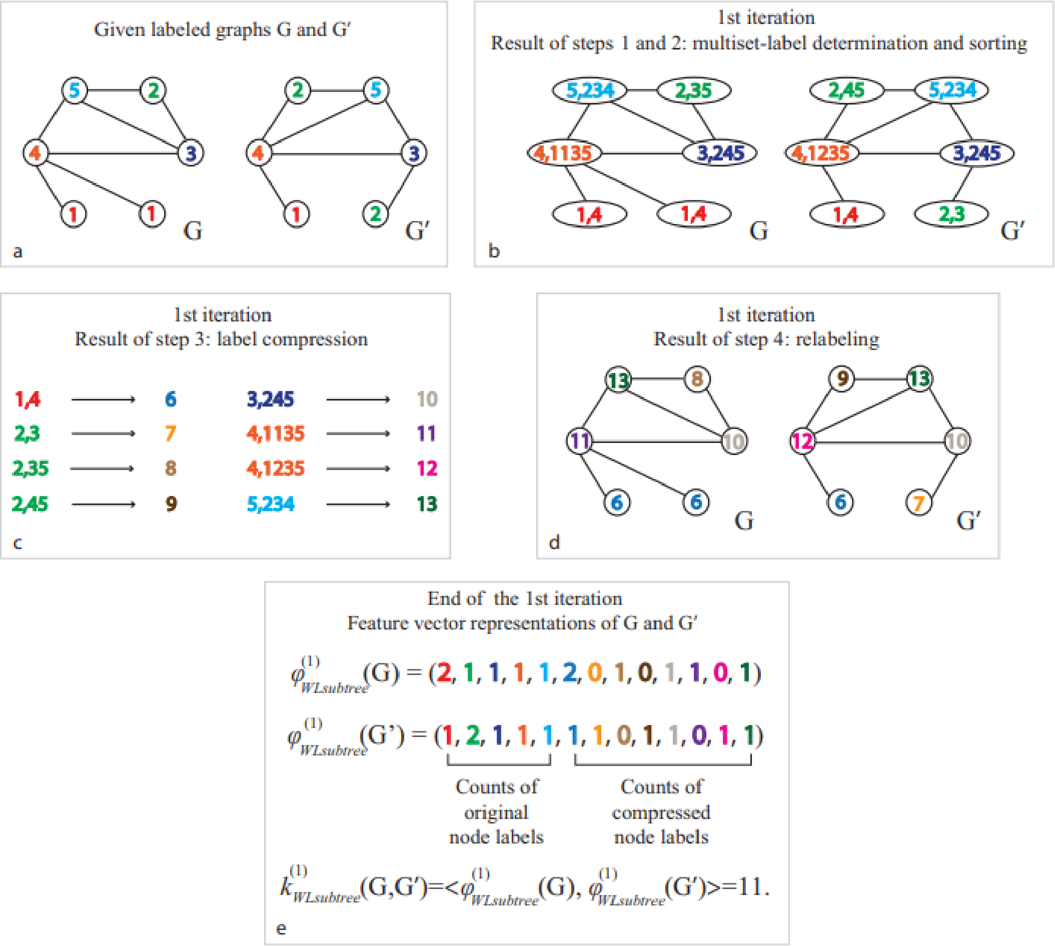
Graph kernel is a kernel function that defines *inner product on graphs* [13]. Intuitively, it is a function that measures the similarity of a pair of two given graphs. There are many different variants of graph kernels, but we'll be focused on the Weisfeiler-Lehman subtree kernel[11]. (Refer to [13] for a extensive study of graph kernels and [12] for a more thorough explanation of the WL subtree kernel)

The WL subtree kernel counts common *original and compressed labels* i.e. common multiset strings in two graphs, resulting from 1-dim WL test. Below shows the algorithm for calculating the WL subtree kernel:

```
Algorithm 2 One iteration of the Weisfeiler-Lehman kernel on $N$ graphs
1: Multiset-label determination
       • Assign a multiset-label $M_h(v)$ to each node $v$ in $G$ which consists of the multiset
         $\{l_{h-1}(u) | u \in \mathcal{N}(v)\}$.
2: Sorting each multiset
       • Sort elements in $M_h(v)$ in ascending order and concatenate them into a string $s_h(v)$.
       • Add $l_{h-1}(v)$ as a prefix to $s_h(v)$.
3: Label compression
       • Map each string $s_h(v)$ to a compressed label using a hash function $g : \Sigma^* \rightarrow \Sigma$ such
         that $g(s_h(v)) = g(s_h(w))$ if and only if $s_h(v) = s_h(w)$.
4: Relabeling
       • Set $l_h(v) := g(s_h(v))$ for all nodes in $G$.
```

Figure 2.2: One iteration of WL subtree kernel

And below figure is a visualization of one run of the WL subtree kernel:



Given labeled graphs G and G′
5
2
4
3
1
1
G
2
5
4
3
1
2
G′
a
1st iteration
Result of steps 1 and 2: multiset-label determination and sorting
5,234
2,35
4,1135
3,245
1,4
1,4
G
2,45
5,234
4,1235
3,245
1,4
2,3
G′
b
1st iteration
Result of step 3: label compression
1,4 → 6
2,3 → 7
2,35 → 8
2,45 → 9
3,245 → 10
4,1135 → 11
4,1235 → 12
5,234 → 13
c
1st iteration
Result of step 4: relabeling
13
8
11
10
6
6
G
9
13
12
10
6
7
G′
d
End of the 1st iteration
Feature vector representations of G and G′
$\varphi^{(1)}_{WLsubtree}(G) = (2, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 0, 1)$
$\varphi^{(1)}_{WLsubtree}(G') = (1, 2, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1)$
Counts of original node labels
Counts of compressed node labels
$k^{(1)}_{WLsubtree}(G,G') = \langle \varphi^{(1)}_{WLsubtree}(G), \varphi^{(1)}_{WLsubtree}(G') \rangle = 11.$
e

Figure 2.3: Illustration of the computation of the WL subtree kernel (with respect to one iteration) for two graphs

2.3 (Overview of) Theoretical Framework

In other words, the WL subtree uses the counts of node labels at different iterations of the WL test as the feature vector of a graph. Intuitively, a node's label at the k-th iteration of the 1-dim WL test represents a **subtree structure of height k rooted at the node**.

Thus, the graph features considered by the WL subtree kernel are essentially counts of different rooted subtrees in the graph!

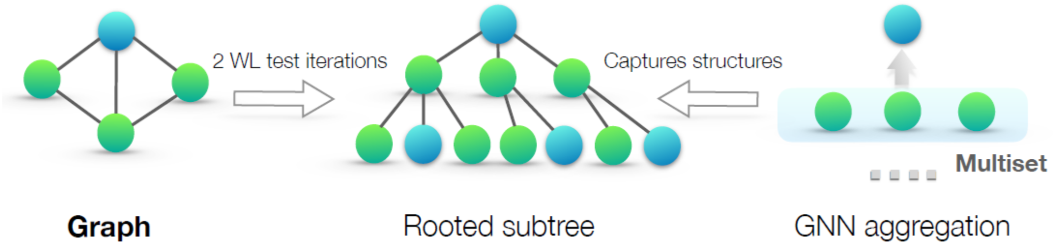


Figure 2.4: An overview of the theoretical framework

What this tells us is that if a GNN's aggregation function captures the *full multiset* of node neighbors, the GNN can capture the rooted subtrees in a recursive manner and be as powerful as the WL test (as shown in above figure).

To analyze the representational power of a GNN, we ask ourselves the question: *when does a GNN map two nodes to the same location in the embedding space)?* **Intuitively, a maximally powerful GNN maps two nodes to the same location only if they have identical subtree structures with identical features on the corresponding nodes.** Since subtree structures are defined recursively via node neighborhoods (above figure), we can reduce our analysis to the question whether a GNN maps two neighborhoods (i.e., two multisets) to the same embedding or representation. A maximally powerful GNN would never map two different neighborhoods, i.e., multisets of feature vectors, to the same representation. This means its aggregation scheme must be injective. Thus, we abstract a GNN's aggregation scheme as a class of functions over multisets that their neural networks can represent, and analyze whether they are able to represent injective multiset functions.

(Above paragraph is taken directly from the paper because it is the key point in this whole work!)

Chapter 3

Main results

3.1 Representational capacity of GNNs

Recall that a maximally powerful GNN has been characterized by an injective aggregation scheme. Thus two (non)isomorphic graphs are mapped to the (different)same representation(s) by a maximally powewful GNN, ideally.

But it turns out that the representational capacity of GNNs is characterized by the WL test, instead of being characterized by Graph Isomorphism.

Lemma 3.1.1
Let G_1 and G_2 be any two non-isomorphic graphs.
If a graph neural network $\mathcal{A} : \mathcal{G} \to \mathbb{R}^d$ maps G_1 and G_2 to different embeddings, the Weisfeiler-Lehman graph isomorphism test also decides G_1 and G_2 are not isomorphic.

The significance of above lemma is that **any aggregation-based GNN is** *at most* **as powerful as the WL test** in distinguishing different graphs.

Now we ask ourselves the question of whether that "bound" is tight. In other words, we want to know if there exists a GNN that is, in principle, as powerful as the WL test in distinguishing different graphs?

And it turns out that is true:

Theorem 3.1.2
Let $\mathcal{A} : \mathcal{G} \to \mathbb{R}^d$ be a GNN.
With a sufficient number of GNN layers, $\mathcal{A}$ maps any G_1 and G_2 that the Weisfeiler-Lehman test of isomorphism decides as non-isomorphic, to different embeddings if the following conditions hold:

- *$\mathcal{A}$ aggregates and updates node features iteratively with*

$$h_v^{(k)} = \phi\left(h_v^{(k-1)}, f\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}_G(v)\right\}\right)\right)$$

  where the functions f, which operates on multisets, and ϕ are injective.

- *$\mathcal{A}$'s graph-level readout, which operates on the multiset of node features $\left\{h_v^{(k)}\right\}$, is injective.*

Before moving on, an important benefit of GNN should be discussed.

Recall that the node feature vectors in the WL test are essentially one-hot encodings, and thus cannot capture similarity between subtrees. It can literally just decide whether two graphs are isomorphic, or not.

GNN (satisfying the criteria in Theorem 3), on the other hand, generalizes the WL test by **learning to embed the subtrees to low-dimensional space**. In other words, GNNs can not only discriminate different structures, but can also **learn to map similar graph structures to similar embeddings and** *capture dependencies between graph structures.* (It was shown in [18] that this is especially helpful for generalization when the co-occurence of subtrees is sparse across different graphs, or there are noisy edges and node features.)

3.2 Graph Isomorphism Network (GIN)

Since we have proved that there really does exist a GNN architecture with the same expressive power as the WL test, let us actually build one! This simple architecture is called the Graph Isomorphism Network, or GIN.

The basic idea is to use deep multisets[20] *i.e.* parametrizing universal multiset functions with neural networks.

Lemma 3.2.1
Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^n$ so that $h(X) = \sum_{x \in X} f(x)$ is unique for each multiset $X \subset \mathcal{X}$ of bounded size.
Moreover, any multiset function g can be decomposed as $g(X) = \phi\left(\sum_{x \in X} f(x)\right)$ for some function ϕ.

Certain popular injective set functions, such as the mean aggregator, are *not* injective **multiset** functions. Above lemma tells us that sum aggregators can represent injective, in fact, universal functions over multisets, and thus we can conceive aggregation schemes that can *represent universal functions over a node and the multiset of its neighbors*, satisfying injectiveness:

Corollary 3.2.1.1
Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^n$ so that for infinitely many choices of ϵ, including all irrational numbers, $h(c, X) = (1 + \epsilon) f(c) + \sum_{x \in X} f(x)$ is unique for each pair (c, X), where $c \in \mathcal{X}$ and $X \subset \mathcal{X}$ is a multiset of bounded size.
Moreover, any function g over such pairs can be decomposed as $g(c, X) = \varphi\left((1 + \epsilon) f(c) + \sum_{x \in X} f(x)\right)$ for some function φ.

Recall the Universal Approximation Theorem[6][5]:

Theorem 3.2.2 (Universal Approximation Theorem (Hornik, 1991))
Define

$$\mathcal{N}_k^{(n)}(\psi) = \left\{ h : \mathbb{R}^k \to \mathbb{R} \,\middle|\, h(x) = \sum_{j=1}^{n} \beta_j \psi(a_j' x - \theta_j) \right\}$$

as the set of all functions implemented by such a network with n hidden units, where ψ is the common activation function of the hidden units.
If ψ is continuous, bounded and nonconstant, then $\mathcal{N}_k^{(n)}(\psi)$ is dense in $\mathscr{C}(X)$ for all compact subsets X of $\mathbb{R}^k$.

This tells us several things:

- Every continuous function can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer.
- The choice of the activation function doesn't matter; it's the multilayer feedforward architecture that gives neural networks the potential of being universal approximators.

Thus we can use MLPs to model and learn f and φ. In practice, $f^{(k+1)} \circ \varphi^{(k)}$ is modeled with one MLP.

From above corollary, we may make ϵ as a learnable parameter, or a fixed scalar. Either way, GIN updates node representations as follows:

$$h_v^{(k)} = \mathrm{MLP}^{(k)} \left(\left(1 + \epsilon^{(k)}\right) h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)} \right)$$

As we've discussed previously, for graph classification tasks, we need a READOUT function. We want to consider all structural information, considering that features from earlier iterations may sometimes generalize better. Using the idea from [17],

$$h_G = \mathrm{CONCAT}\left(\mathrm{READOUT}\left(\left\{h_v^k | v \in V(G)\right\}\right) \middle| k = 0, 1, \ldots, K\right)$$

i.e. information from *all* depths/iterations of the model is used.

3.3 Less powerful, but still interesting GNNs

Here, we consider GNNs that do not satisfy the conditions as described in Theorem 3 ,and GNNs with different choice of AGGREGATE (Max-pooling, Mean)

Some of the questions that will be answer here are:

- What if 1-layer perceptron is used instead of MLPs?
- What if the sum $h(X) = \sum_{x \in X} f(x)$ is replaced by mean/max pooling?

(For just a summary, skip to the last subsection)

3.3.1 1-Layer Perceptrons

The function f in Lemma 3.2.1 helps map distinct multisets to unique embeddings. It was shown by the Universal Approximation Theorem[5] that f can be parametrized by MLPs.

Many modern GNNs, however, use a *1-layer perceptron* $\sigma \circ W$: a linear mapping followed by a non-linear activation function. So the natural question to ask is "Is 1-layer perceptron enough for graph learning?"

And the answer is, as expected, no.

Lemma 3.3.1
There exist finite multisets $X_1 \neq X_2$ so that for any linear mapping W, $\sum_{x \in X_1} \mathrm{ReLU}(Wx) = \sum_{x \in X_2} \mathrm{ReLU}(Wx)$

Above lemma shows that unlike models using MLPs, 1-layer perceptron (even with the bias term) is not a universal approximator of multiset functions.

3.3.2 GCN

Recall that GCN[7] takes the form:

$$h_v^{(k)} = \mathrm{ReLU}\left(W \, \mathrm{MEAN}\left\{h_u^{(k-1)} : u \in \mathcal{N}_G(v) \cup \{v\}\right\}\right)$$

i.e. it utilizes *mean aggregator.*

Now consider two multisets: $X_1 = (S, m)$ and $X_2 = (S, km)$ It can be easily observed that any mean aggregator maps X_1 and X_2 to the *same* embeddings! Generally speaking, **mean aggregator captures the** *distribution of elements in a multiset.*

Corollary 3.3.1.1
Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^n$ so that $h(X) = \frac{1}{|X|} \sum_{x \in X} f(x)$, $h(X_1) = h(X_2)$ if only if multisets X_1 and X_2 have the same distribution. That is, assuming $|X_2| \geq |X_1|$, we have $X_1 = (S, m)$ and $X_2 = (S, km)$ for some $k \in \mathbb{N}$.

(This is as powerful as the sum aggregator if the node features are diverse and rarely repeat, and thus *effective for node classification.*)

3.3.3 GraphSAGE

Recall that GraphSAGE[4] takes the form:

$$a_v^{(k)} = \mathrm{MAX}\left(\left\{\mathrm{ReLU}\left(W h_u^{(k-1)}\right) : u \in \mathcal{N}_G(v)\right\}\right)$$

$$h_v^{(k)} = W\left[h_v^{(k-1)}, a_v^{(k)}\right]$$

i.e. it utilizes max-pooling aggregator.

Unlike previous aggregators, max-pooling can't capture exact structure nor the distribution! What it can capture is the **underlying set of muliset** i.e. S in $X = (S, m)$

Corollary 3.3.1.2
Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^\infty$ so that $h(X) = \max_{x \in X} f(x)$, $h(X_1) = h(X_2)$ if only if multisets X_1 and X_2 have the same underlying set.

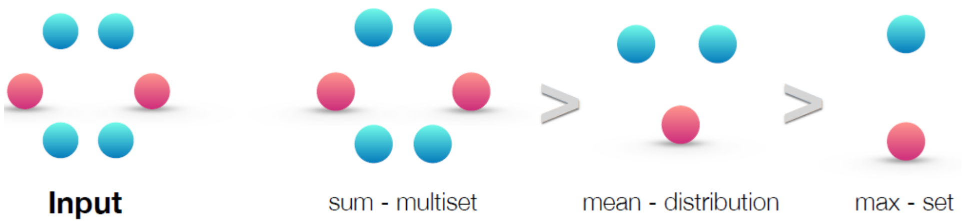
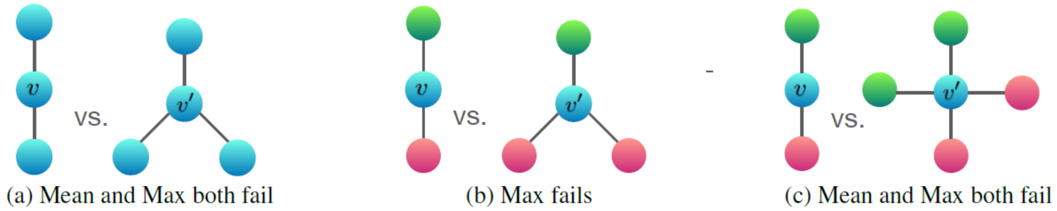


Figure 3.1: Ranking by expressive power for sum, mean, and max aggregators over a multiset



(a) Mean and Max both fail

(b) Max fails

(c) Mean and Max both fail

Figure 3.2: Examples of graph structures that mean and max aggregators fail to distinguish

3.3.4 Summary

From above discussions, we can now rank the three aggregators by their representational power:

- Sum over multiset aggregator (as in GIN) completely captures the exact structure of graph.
- Mean aggregator (as in GCN) captures the statistical and distributional information of the graph.
- Max-pooling aggregator (as in GraphSAGE) captures the representative elements of the graph, or its skeleton

Chapter 4

Experiments

The goal of the experiment is to compare the training and test performance of GIN and (less powerful) GNN variants.[1] Specifically,

- Training set performance: compare different GNN models based on their *representational power*
- Test set performance: quantifies *generalization ability*

4.1 Experiment Design

9 graph classification benchmarks[18] were used:

- 4 bioinformatics datasets: MUTAG, PTC, NCI1, PROTEINS
- 5 social network datasets: COLLAB, IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI5K

The authors have preprocessed some of the datasets. The nodes of bioinformatics datasets already have categorical input features, so they were left untouched. Social network datasets, on the other hand, have no node features. To not allow the models to rely on the input node features but mainly learn from the network structure, the authors have created the node features for the social network datasets as follows: for the REDDIT datasets, all node feature vectors were set to be the same; for the other social graphs, one-hot encodings of node degrees were used.

Several models were used:

- GIN $-\epsilon$: GIN that *learns* ϵ by gradient descent
- GIN -0: GIN that fixes ϵ to 0.
- Architectures that replace the sum in the GIN -0 aggregation with mean or max-pooling, or replace MLPs with 1-layer perceptrons
- GCN
- GraphSAGE

(There exists some contrived graphs that GIN $-\epsilon$ can distinguish but GIN cannot.) (Specific choices of hyperparameters are omitted in this writing. Curious readers may refer to the original paper[15].)

Baselines that were used:

- WL subtree kernel with C-SVM used as a classifier
- Deep learning architectures i.e. DCNN, PATCHY-SAN, DGCNN
- AWL

[1] The code is available at `https://github.com/weihua916/powerful-gnns`

4.2 Results / Analysis

4.2.1 Training set performance

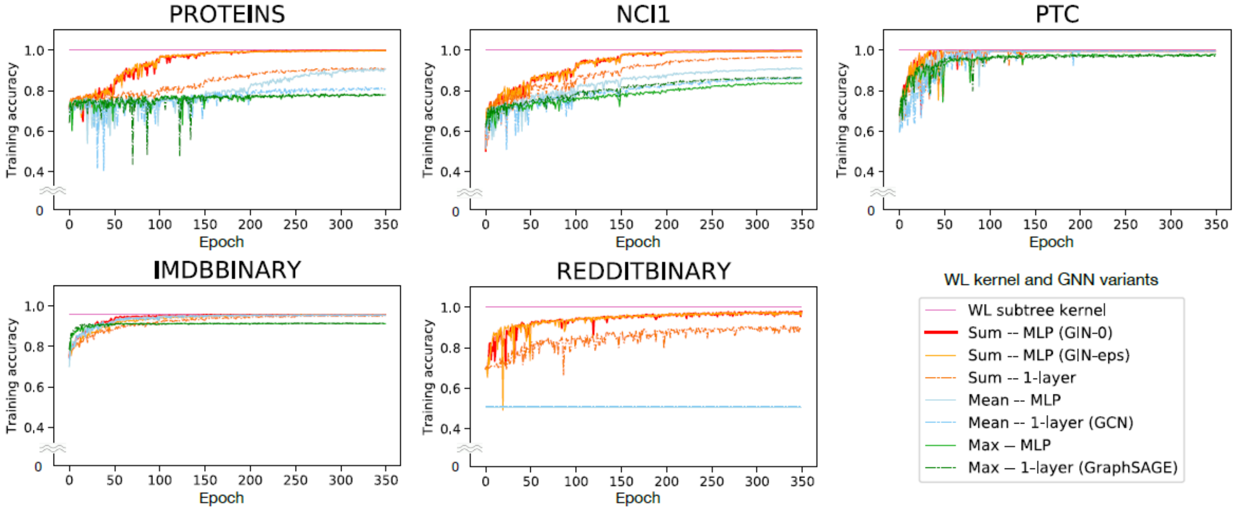


Figure 4.1: Training set performance

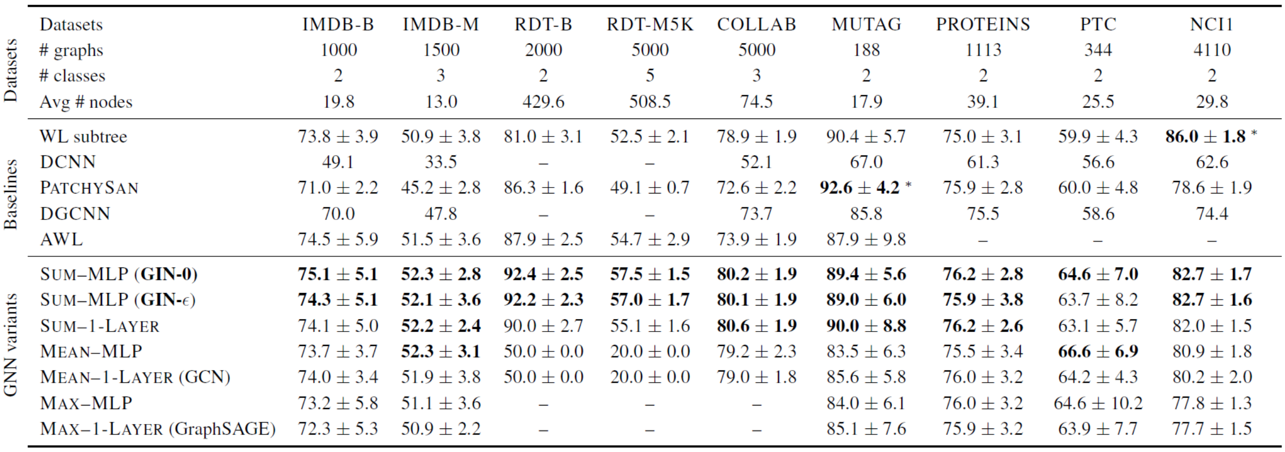
Above figure shows the training curves of GINs and less powerful GNN variants with the same hyper-parameter settings.

First, observe that both the theoretically most powerful GNN, i.e. GIN $-\epsilon$ and GIN -0, are able to almost perfectly fit all the training sets. In comparison, the GNN variants using mean/max pooling or 1-layer perceptrons severely underfit on many datasets. This shows that the training accuracy patterns align with the ranking of the models' representational power.

Also, the training accuracies of the GNNs never exceed those of the WL subtree kernel. This aligns with the result that the WL test provides an upper bound for the representational capacity of the aggregation-based GNNs.

One particular interesting observation is that the explicit learning of ϵ in GIN $-\epsilon$ yielded no gain in fitting training data compared to GIN -0, where ϵ is fixed to 0.

4.2.2 Test set performance



	Datasets	IMDB-B	IMDB-M	RDT-B	RDT-M5K	COLLAB	MUTAG	PROTEINS	PTC	NCI1
Datasets	# graphs	1000	1500	2000	5000	5000	188	1113	344	4110
	# classes	2	3	2	5	3	2	2	2	2
	Avg # nodes	19.8	13.0	429.6	508.5	74.5	17.9	39.1	25.5	29.8
Baselines	WL subtree	73.8 ± 3.9	50.9 ± 3.8	81.0 ± 3.1	52.5 ± 2.1	78.9 ± 1.9	90.4 ± 5.7	75.0 ± 3.1	59.9 ± 4.3	**86.0 ± 1.8** *
	DCNN	49.1	33.5	–	–	52.1	67.0	61.3	56.6	62.6
	PATCHYSAN	71.0 ± 2.2	45.2 ± 2.8	86.3 ± 1.6	49.1 ± 0.7	72.6 ± 2.2	**92.6 ± 4.2** *	75.9 ± 2.8	60.0 ± 4.8	78.6 ± 1.9
	DGCNN	70.0	47.8	–	–	73.7	85.8	75.5	58.6	74.4
	AWL	74.5 ± 5.9	51.5 ± 3.6	87.9 ± 2.5	54.7 ± 2.9	73.9 ± 1.9	87.9 ± 9.8	–	–	–
GNN variants	SUM–MLP (**GIN-0**)	**75.1 ± 5.1**	**52.3 ± 2.8**	**92.4 ± 2.5**	**57.5 ± 1.5**	**80.2 ± 1.9**	**89.4 ± 5.6**	**76.2 ± 2.8**	**64.6 ± 7.0**	**82.7 ± 1.7**
	SUM–MLP (**GIN-ϵ**)	**74.3 ± 5.1**	**52.1 ± 3.6**	**92.2 ± 2.3**	**57.0 ± 1.7**	**80.1 ± 1.9**	**89.0 ± 6.0**	**75.9 ± 3.8**	63.7 ± 8.2	**82.7 ± 1.6**
	SUM–1-LAYER	74.1 ± 5.0	**52.2 ± 2.4**	90.0 ± 2.7	55.1 ± 1.6	**80.6 ± 1.9**	**90.0 ± 8.8**	**76.2 ± 2.6**	63.1 ± 5.7	82.0 ± 1.5
	MEAN–MLP	73.7 ± 3.7	**52.3 ± 3.1**	50.0 ± 0.0	20.0 ± 0.0	79.2 ± 2.3	83.5 ± 6.3	75.5 ± 3.4	**66.6 ± 6.9**	80.9 ± 1.8
	MEAN–1-LAYER (GCN)	74.0 ± 3.4	51.9 ± 3.8	50.0 ± 0.0	20.0 ± 0.0	79.0 ± 1.8	85.6 ± 5.8	76.0 ± 3.2	64.2 ± 4.3	80.2 ± 2.0
	MAX–MLP	73.2 ± 5.8	51.1 ± 3.6	–	–	–	84.0 ± 6.1	76.0 ± 3.2	64.6 ± 10.2	77.8 ± 1.3
	MAX–1-LAYER (GraphSAGE)	72.3 ± 5.3	50.9 ± 2.2	–	–	–	85.1 ± 7.6	75.9 ± 3.2	63.9 ± 7.7	77.7 ± 1.5

Figure 4.2: Test set performance

Above table compares the test accuracies of GINs (Sum–MLP), other GNN variants, as well as the state-of-the-art baselines.

First, GINs, especially GIN -0, outperform the less powerful GNN variants on all the 9 datasets, achieving state-of-the-art performance.

In the Reddit datasets, where all nodes share the same scalar as node feature, GINs and sum-aggregation GNNs accurately capture the graph structure and significantly outperform other models. Mean-aggregation GNNs, however, fail to capture any structures of the unlabeled graphs and do not perform better than random guessing. (The same phenomenon is observed when the node degrees are provided as input features.) This, again, aligns with the ranking of the models' representational power, based on its aggregator.

One interesting observation here is that GIN -0 slightly but consistently outperforms GIN $-\epsilon$. The authors explain this better generalization by arguing that the model complexity of GIN $-\epsilon$ is higher than GIN -0. This has not been theoretically confirmed, though.

One might ask why WL subtree kernel doesn't act as an upper bound as it did in the training set performance. This can be explained by the fact that WL kernel, unlike the GINs, can't learn how to combine node features, which might be quite informative for a given prediction task.

4.3 Remark

Note that the authors have not performed experiments regarding node classification. As they've stated in `https://openreview.net/forum?id=ryGs6iA5Km`, there are two reasons of why they didn't do that.

Borrowing their words,

1. In many node classification applications, node features are rich and diverse (e.g. bag-of-words representation of papers in a citation network), so GNN models like GCN and GraphSAGE are often already able to fit the training data well.

2. Many node classification tasks assume limited training labels (semi-supervised learning scenario); thus, the inductive bias of GNNs also plays a key role in empirical performance. For example, the statistical and distributional information of neighborhood features may provide a strong signal for many node classification tasks.

It may be the case that GIN performs well on node classification tasks. However the performance on node classification tasks are less directly explained by the authors' theory of representational power, which is why the authors have left that part out.

Chapter 5

Conclusion

5.1 Summary

In summary,

- Theoretical foundations for reasoning about the expressive power of GNNS
- Tight bounds on the representational capacity of popular GNN variants. (cf. WL test)
- Designed a provably maximally powerful GNN under the neighborhood aggregation framework (*Graph Isomorphism Network*)

5.2 Future research

Here are some possible directions for future research:

- Different aggregators?
- Go beyond neighborhood aggregation
- Understand/improve the generalization properties of GNNs
- What if the node features are continuous (uncountable)?
- Better understanding of the optimization landscape

Bibliography

[1] László Babai. “Graph isomorphism in quasipolynomial time [extended abstract]”. In: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016.* 2016, pp. 684–697.

[2] László Babai and Ludek Kucera. “Canonical Labelling of Graphs in Linear Average Time”. In: *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979.* 1979, pp. 39–46.

[3] Jin-yi Cai, Martin Fürer, and Neil Immerman. “An optimal lower bound on the number of variables for graph identifications”. In: *Combinatorica* 12.4 (1992), pp. 389–410.

[4] William L. Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 2017, pp. 1024–1034.

[5] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4.2 (1991), pp. 251–257.

[6] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366.

[7] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* 2017.

[8] Tao Lei et al. “Deriving Neural Architectures from Sequence and Graph Kernels”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017.* 2017, pp. 2024–2033.

[9] Franco Scarselli et al. “Computational Capabilities of Graph Neural Networks”. In: *IEEE Trans. Neural Networks* 20.1 (2009), pp. 81–102.

[10] Franco Scarselli et al. “The Graph Neural Network Model”. In: *IEEE Trans. Neural Networks* 20.1 (2009), pp. 61–80.

[11] Nino Shervashidze and Karsten M. Borgwardt. “Fast subtree kernels on graphs”. In: *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.* 2009, pp. 1660–1668.

[12] Nino Shervashidze et al. “Weisfeiler-Lehman Graph Kernels”. In: *J. Mach. Learn. Res.* 12 (2011), pp. 2539–2561.

[13] S. V. N. Vishwanathan et al. “Graph Kernels”. In: *J. Mach. Learn. Res.* 11 (2010), pp. 1201–1242.

[14] Boris Weisfeiler and Andrei A. Lehman. “A reduction of a graph to a canonical form and an algebra arising during this reduction”. In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968), pp. 12–16.

[15] Zonghan Wu et al. “A Comprehensive Survey on Graph Neural Networks”. In: *arXiv e-prints* (Jan. 2019). arXiv: `1901.00596 [cs.LG]`.

[16] Keyulu Xu et al. “How Powerful are Graph Neural Networks?” In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.* 2019.

[17] Keyulu Xu et al. “Representation Learning on Graphs with Jumping Knowledge Networks”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018, pp. 5449–5458.

[18] Pinar Yanardag and S. V. N. Vishwanathan. “Deep Graph Kernels”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*. 2015, pp. 1365–1374.

[19] Zhitao Ying et al. “Hierarchical Graph Representation Learning with Differentiable Pooling”. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.* 2018, pp. 4805–4815.

[20] Manzil Zaheer et al. “Deep Sets”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 2017, pp. 3391–3401.

[21] Muhan Zhang et al. “An End-to-End Deep Learning Architecture for Graph Classification”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018.* 2018, pp. 4438–4445.