How Powerful are GNNs?

Report

Team 29

Surabhi Jain	(2021121004)
Shruti Kolachana	(2020102053)
Maulesh Gandhi	(2020112009)
Deepthi Chandak	(2020102013)

Introduction

A set of objects, and the connections between them, are naturally expressed as a *graph.*

A graph is a data structure made up of two elements: vertices and edges. A graph is often defined as G=(V, E), where V is a set of nodes and E. An Adjacency matrix, A, is commonly used to depict a graph. If a graph has N nodes, then A has an NxN dimension.

To further describe each node, edge, or the entire graph, we can store information in each piece of the graph as vertex(or node) embedding, edge(or link) attributes, or embedding and global embedding for the entire graph.

Various data types can be modeled as a graph, such as images, text, social networks, etc.

1. Images as graphs

We can think of images as graphs where each pixel represents a node and is connected through an edge to adjacent pixels. Each non-border pixel has 8 neighbors, and the information stored at each node is a 3-dimensional vector representing the pixel's RGB value.

2. Text as graphs:

We can represent a text by giving indices to the characters, words, or tokens and representing the text as a sequence of these indices. This will result in a simple directed graph where each character or index is a node connected via an edge to the node that follows it.

3. Molecules as graphs
4. Social networks as graphs

Definitions

Let G = (V, E) denote a graph with node feature vectors X_v for v ∈ V.

Node classification Task:

Each node v ∈ V has an associated label y_v , and the goal is
to learn a representation vector h_v of v such that v’s label can be predicted as $y_v = f(h_v)$

Graph classification Task:

Given a set of graphs $G_1, ..., G_N \subseteq G$ and their labels $y_1, ..., y_N \subseteq Y$, we
aim to learn a representation vector h_G that helps predict the label of an entire graph, $y_G = g(h_G)$.

GNN:

A GNN is an optimizable transformation on all graph attributes (nodes, edges, global context) that preserves graph symmetries (permutation invariances).

GNNs follow a neighborhood aggregation scheme, where the representation vector of a node is computed by recursively aggregating and transforming representation vectors of its neighboring nodes.
[https://arxiv.org/abs/1810.00826]

GNNs learn a representation vector of a node, h_v, or the entire graph, h_G, using the node features and the graph structure.

$$a_v^{(k)} = AGGREGATE^{(k)}(h_u^{(k-1)} : u \in N(v))$$

$$h_v^{(k)} = COMBINE^{(k)}(h_v^{(k-1)}, a_v^k)$$

Here, $h_v^{(k)}$ is the feature vector of node v at the kth iteration/layer,$h_v^{(0)} = X_v$ and $N(v)$ is the set of nodes adjacent to v.

Graph Sage:

In the pooling variant of graphSAGE,element-wise max-pooling is used.

$$a_v^{(k)} = MAX(ReLU(W * h_u^{(k-1)} : u \in N(v))$$

MAX represents an element-wise max-pooling

The COMBINE step could be a concatenation followed by a linear mapping $W * [h_u^{(k-1)}, a_v^{(k)}]$

GCN:

In GCN,element-wise mean pooling is used

AGGREGATE and COMBINE steps are integrated as:

$$h_v^{(k)} = ReLU(W * MEAN(h_u^{(k-1)} : u \in N(v) \cup \{u\})$$

Node classification: For node classification, the node representation $h_v^{(k)}$ of the final iteration is used for prediction.

Graph classification: Readout function is used for the final prediction

(the READOUT function aggregates node features from the final iteration to
obtain the entire graph's representation h_G:

$$h_v^{(k)} = READOUT(\{h_v^{(k)} : v \in G\})$$

Many GNN variants have been proposed and have achieved good results on both node and graph classification tasks.

There is limited understanding of their representational properties and limitations. In the original paper,the authors have presented a theoretical framework for analyzing the expressive power of GNNs to capture different graph structures. We validate their results by training and comparing the different GNN variants on 3 benchmarking datasets.

The Objectives/main results summary from Paper[https://arxiv.org/abs/1810.00826]]

1.GNNs are at most as powerful as the WL test in distinguishing graph structures.
2. Developed a simple neural architecture, Graph Isomorphism Network (GIN), and showed that its discriminative/representational power is equal to the power of the WL test.

3.GIN is maximally powerful GNN

WL(Weisfeiler-Lehman) test

Two graphs are considered isomorphic a pair of nodes may be connected by an edge in the first graph if and only if the corresponding pair of nodes in the second graph is also connected by an edge in the same way.

It produces a canonical form for each graph. If the canonical forms of two graphs are not equivalent, then the graphs are definitively not isomorphic. However, two non-isomorphic graphs can have the same canonical form, so we cannot conclude that two graphs are isomorphic just by this test.

The Algorithm:

For iteration, assign a tuple $L_{i,n}$ to each node containing the node's old compressed label and a multiset of the node's neighbors' compressed labels. Also at each iteration assign to each node a new "compressed" label $C_{i,n}$ for that node's set of labels. Any two nodes with the same Li, n will get the same compressed label.

1. To start, initialize $C_{i,n} = 0$ for all nodes
2. At iteration i of the algorithm (beginning with i=0), for each node set $L_{i,n}$ to be a tuple containing the node's old label $C_{i-1,n}$ and the multiset of compressed node labels $C_{i-1,m}$ from all m nodes neighboring n from the previous iteration i-1.
3. Complete iteration i by setting $C_{i,n}$ to be a new "compressed" label, such as a hash of $L_{i,n}$. Any two nodes with the same labels must get the same compressed label .
4. Partition the nodes in the graph by their compressed label. Repeat 2 and 3 for up to N (the number of nodes) iterations, or until there is no change in the partition of nodes by compressed label from one iteration to the next.

GIN

Graph Isomorphism Network (GIN) satisfies the conditions in Theorem 3.

Theorem 3. Let A: $G \rightarrow R^d$ be a GNN. With a sufficient number of GNN layers, A maps any graphs G1 and G2 that the Weisfeiler-Lehman test of isomorphism decides as non-isomorphic, to different embeddings if the following conditions hold:

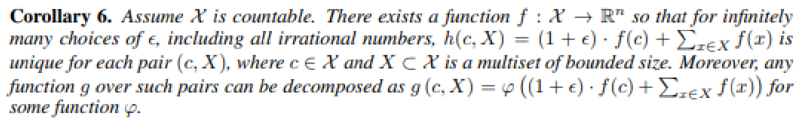
a) A aggregates and updates node features iteratively with

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

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

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

It generalizes the WL test and achieves maximum discriminative power among GNNs. This new aggregator has to provide distinct node embeddings when working with non-isomorphic networks to be as effective as the WL test. They provide a corollary that gives a simple and concrete formulation among many such aggregation schemes.



Corollary 6. *Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \rightarrow \mathbb{R}^n$ so that for infinitely many choices of ϵ, including all irrational numbers, $h(c, X) = (1+\epsilon) \cdot f(c) + \sum_{x \in X} f(x)$ 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 $g(c, X) = \varphi\left((1+\epsilon) \cdot f(c) + \sum_{x \in X} f(x)\right)$ for some function φ.*

Since the Universal Approximation Theorem states that a neural network with 1 hidden layer can approximate any continuous function for inputs within a specific range, the authors propose a method that involves learning the two injective functions using an MLP (Multilayer Perceptron). Because MLPs can depict the composition of functions, we simulate both injective functions using a single MLP.

A particular node's hidden vector may be determined using the following formula:

$$h_i = MLP((1+\epsilon) * x_i + \sum_{j \in N_i} x_j)$$

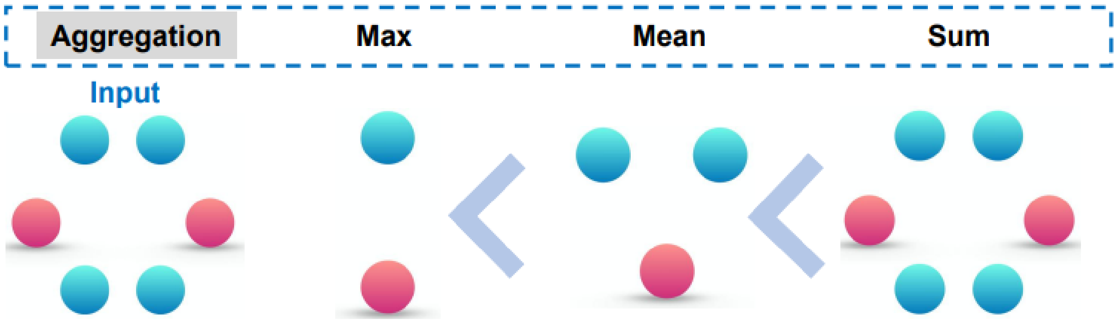
In this formula,ε determines the target node's value about its neighbors (it has equal importance if ε = 0). It might be a fixed scalar or a learnable parameter.

In general, there could be a lot of other strong GNNs. GIN is one of the primary, maximally effective GNN examples.

It is important to note that 1-layer perceptrons are not enough for graph learning since they are not universal approximators of multiset functions, even with a bias term.

GRAPH-LEVEL READOUT OF GIN

Graph-level readout, also known as global pooling, is the process of creating a graph embedding from the node. A simple way to obtain a graph embedding is to use the mean, sum, or max of every node embedding h_i. The expressive power of sum, mean, and max aggregators over a multiset is ranked as:



[src: https://arxiv.org/abs/1810.00826]

When the "skeleton" or representative pieces are crucial, max aggregation works effectively.

Whenever statistical and distributional data are crucial, mean aggregation works well.

Sum aggregation performs well for all common situations.

Also, the number of iterations can significantly impact the updated features. Ideally, structures should get more refined till a sufficient number of iterations. However, occasionally features from earlier iterations might generalize better. To overcome this issue, node embeddings are added together for each layer, and the outcome is concatenated. The expressiveness of the sum operator and the concatenation's ability to remember prior iterations are combined in this method. The graph-level readout obtained from this can be defined as:

$$h_G = \text{CONCAT}\Big(\text{READOUT}\Big(\Big\{h_v^{(k)} | v \in G\Big\}\Big) \mid k = 0, 1, \ldots, K\Big).$$

Study on GNN

To understand better the popular GNNs, the authors perform an ablation study on aggregation: (1) 1-layer perceptrons instead of MLPs and (2) mean or max-pooling instead of the sum. They show that while GCN can perform well on node classification, GNNs like GCN and GraphSAGE are less powerful than the WL test.

1-Layer Perceptrons are not Sufficient

GNNs with 1-layer perceptrons can embed different graphs to different locations to some degree, such embeddings may not adequately capture structural similarity, and can be difficult for simple classifiers, e.g., linear classifiers, to fit.

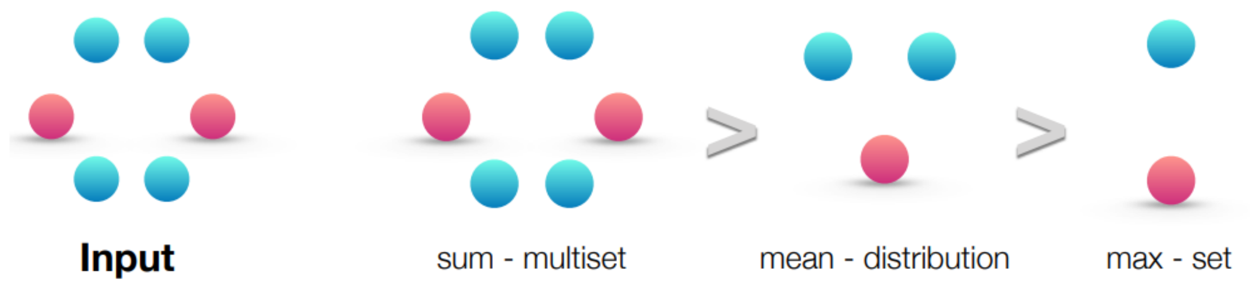
The authors prove that there exists multisets $X_1 \neq X_2$ so that for any linear mapping, $\sum_{x \in X_1} \text{ReLU}(Wx) = \sum_{x \in X_2} \text{ReLU}(Wx)$

Counter Example: Consider $X_1 \neq X_2$ consisting of scalars that are of the same sign (either all positive or all nonnegative) and $\sum_{x \in X_1} x = \sum_{x \in X_2} x$. Then by nonlinearity, we have $\text{ReLU}(\sum_{x \in X_1} Wx) = \text{ReLU}(\sum_{x \in X_2} Wx)$. Hence the aggregator is not an injective function on multisets.

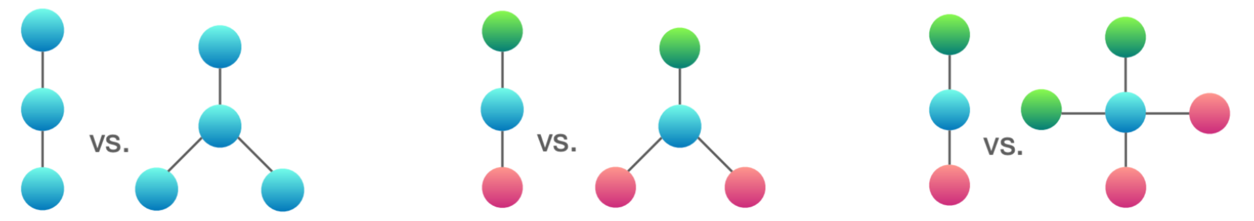
Note that the original GCN does not have a biased term and with a biased term, things might be better. Meanwhile, we need at least one hidden layer for Universal Approximation Theorem to hold. The authors claim that this may result in an underfitting model that cannot well capture the structural information.

Structures that Confuse Mean and Max-Pooling

Two common aggregation methods are mean pooling and max pooling, as they are invariant under node permutation. However, these methods can fail on some regular graphs, particularly when the multiplicity/number of occurrences/counting of node features also matters.



In the toy binary graph classification example I tried, if you do not give node degrees as node features at the beginning, you will not be able to even distinguish cycles from stars.



(a) Mean and Max both fail (b) Max fails (c) Mean and Max both fail

[src: https://arxiv.org/abs/1810.00826]

Experiments

We evaluated and compare the training and test performance of GIN,GraphSAGE with max pooling and GCN with mean pooling on the following three datasets:

Datasets used:

PROTEINS

graphs-1113

classes-2

avg nodes-39.06

edges-72.82

MUTAG

graphs-118

classes-2

avg nodes-17.93

edges-19.97

<u>NC1</u>

graphs-4110

classes-2

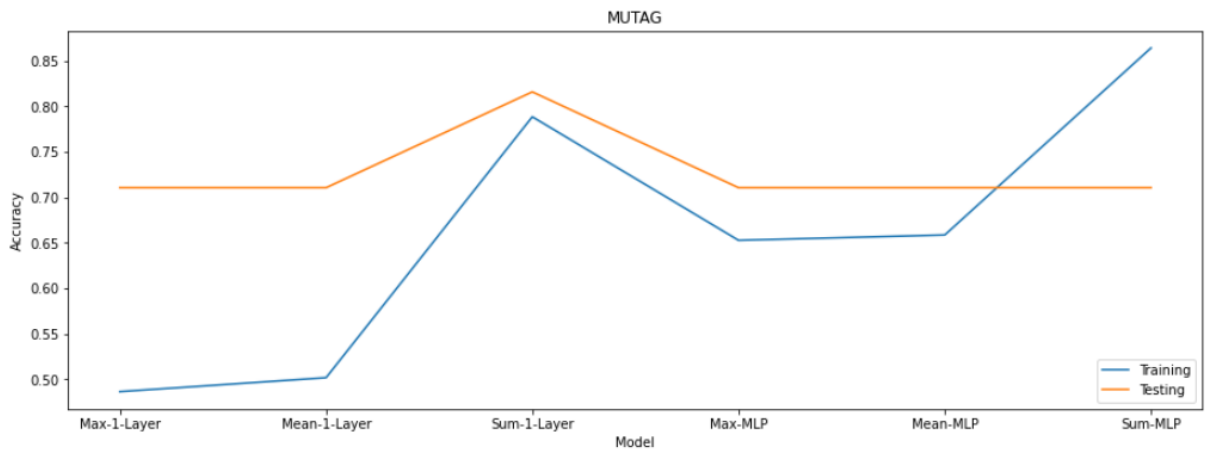
avg nodes-29.87

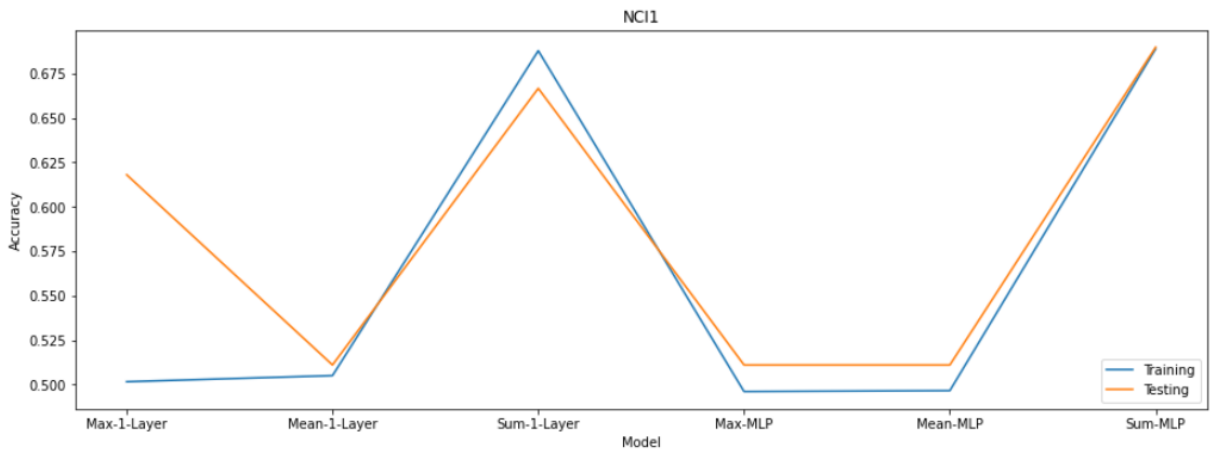
edges-32.30

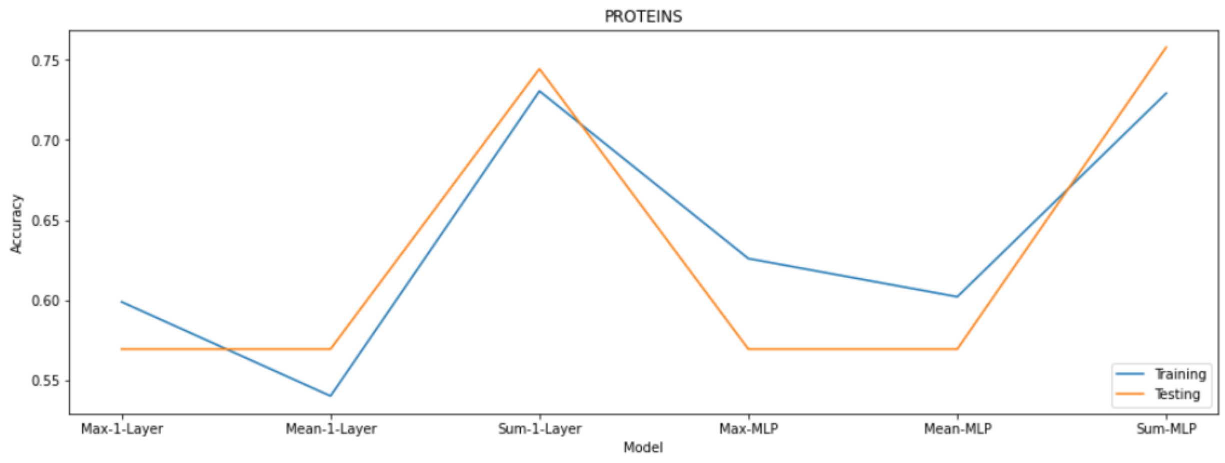
We have considered the GIN variant where epsilon=0

We compare different GNN models using the training set performance which quantifies their representational power. Test set performance quantifies generalization ability.

Batch-size = 64







Training Accuracies:

Dataset	Max-1-Layer	Max-MLP	Mean-1-Layer	Mean-MLP	Sum-1-Layer	Sum-MLP (GIN)
MUTAG	`48.65%`	`65.27%`	`50.18%`	`65.86%`	`78.84%`	`86.40%`
NCI1	`50.15%`	`49.6%`	`50.5%`	`49.65%`	`68.78%`	`68.89%`
PROTEINS	`59.89%`	`62.61%`	`54.02%`	`60.22%`	`73.05%`	`72.91%`

Testing Accuracies:

Dataset	Max-1-Layer	Max-MLP	Mean-1-Layer	Mean-MLP	Sum-1-Layer	Sum-MLP (GIN)
MUTAG	`71.05%`	`71.05%`	`71.05%`	`71.05%`	`81.58%`	`71.05%`
NCI1	`61.80%`	`51.09%`	`51.09%`	`51.09%`	`66.66%`	`68.98%`

PROTEINS	`56.95%`	`56.95%`	`56.95%`	`56.95%`	`74.44%`	`75.78%`

Results

We validated the theoretical analysis of the representational power of GNNs(https://arxiv.org/abs/1810.00826) by comparing their training and testing accuracies.

Models with higher representational power should have higher training set accuracy.

GIN gives the highest accuracies in all the datasets considered which thus validates the theory that GIN has the maximum representational power among all GNN variants.

Conclusion

We did experiments to compare the expressive power of
GNNs, and validated the theoretical analysis of the representational power of GNNs(https://arxiv.org/abs/1810.00826) including the maximally powerful GNN introduced by the original authors by comparing their training and test accuracies.

References:

- https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/
- Original Paper: https://arxiv.org/abs/1810.00826
- [1609.02907] Semi-Supervised Classification with Graph Convolutional Networks (arxiv.org)
- [1706.02216] Inductive Representation Learning on Large Graphs (arxiv.org)
- weihua916/powerful-gnns: How Powerful are Graph Neural Networks? (github.com)

Contributions:

- Surabhi Jain & Shruti Kolachana: GNN & GIN models
- Maulesh Gandhi & Deepthi Chandak: Datasets & training-testing