How Powerful Are Graph Neural Networks

- **Eswara Rohan (2020102002)**
- **Dheeraj Murugan Sai (2020102020)**
- **K. Sesha Sarath (2020102028)**
- **S. Deepak (2020102063)**

Abstract

Graph Neural Networks (GNNs) are an effective framework for representation learning of graphs. GNNs follow a neighborhood aggregation scheme for node and graph classification tasks. In this project, we study discriminative power of popular GNN variants such as Convolutional Networks and GraphSAGE; and analyze their expressive power to capture different graph structures.

Introduction

In real life we use GNNs for effective representation and classification of graphs and nodes. GNNs follow a recursive neighborhood aggregation scheme, where each node aggregates feature vectors of its neighbors to compute its new feature vector. After K iterations of aggregation, feature vector of the node captures the structural information within the node's K-hop neighborhood.

A powerful test for graph classification, Weisfeiler-Lehman (WL) graph isomorphism test has close connection with GNNs. GNN can have as large discriminative power as the WL test if the GNN's aggregation scheme is highly expressive and can model injective functions.

Graph Terminology

G = (V,E) - Graph with vertices V and edges E

X_v - Feature Vectors Of Nodes for $v \epsilon V$

Classification Types -

1. Node Classification -

   In node classification, each node has an associated label y_v , representation vector h_v such that $y_v = f(h_v)$

2. Graph Classification -

   In graph classification, we have set of graphs $G_1, G_2, \ldots, G_N \subseteq G$ and their associated labels $y_1, y_2, \ldots, y_N \subseteq Y$, representation vector h_G that helps to predict the label of entire graph $y_G = g(h_G)$

General Approach In GNN -

As we know in Convolutional Neural Networks, the pixel value in an image gets updated with a function which involves its neighboring pixels in the convolution layer, here in Graph Neural Networks we follow a similar approach where we use AGGREGATE and COMBINE operations on each node for K iterations (GNN with K layers).

Modern GNNs follow a neighborhood aggregation scheme where we iteratively update the representation of a node by aggregating representations of its neighbors. The function used for aggregating neighbors is AGGREGATE and the one used for updating the representation vector is COMBINE.

Formally, kth layer of GNN is -

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

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

where, $h_v^{(k)}$ is the feature vector of node v at k^{th} iteration/layer. We initialize $h_v^{(0)} \ = \ X_v$ and $N(v)$ is a set of nodes adjacent to v.

We can use max pooling, mean, sum,... functions for AGGREGATE and linear combination,.... for COMBINE. The choice of AGGREGATE and COMBINE functions are very crucial as it may affect the accuracy of the model.

GCN and GraphSAGE -

In this project, we compare our results with GCN and GraphSAGE in which the aggregation functions are MEAN and MAX pooling respectively.

GraphSAGE -

AGGREGATE - $a_v^{(k)} \ = \ MAX(ReLU(W.h_u^{(k-1)}), \forall \ u \epsilon \ N(u))$

COMBINE - Concatenation followed by a linear mapping W.

GCN-

The AGGREGATE and COMBINE steps in GCN are integrated as follows:

$$h_v^{(k)} \ = \ ReLU(W.MEAN\{h_u^{(k-1)}, \forall \ u \ \epsilon \ N(v) \ \cup \ v\})$$

For node classification, the node representation h_v^k of the final iteration is used for prediction whereas for graph classification, the READOUT function aggregates node features from the final iteration to obtain the entire graph's representation h_G.

$$h_G \ = \ READOUT(h_v^{(k)} | \ v \epsilon G) \ .$$

Weisfeiler-Lehman test (WL Test): -

A powerful test known to distinguish a broad class of graphs and have a close connection with GNN is Weisfeiler-Lehman (WL) graph isomorphism test. WL test iteratively aggregates the labels of nodes and their neighborhoods and hashes the aggregated labels into unique new labels.

The algorithm decides that two graphs are non-isomorphic if at some iteration the labels of the nodes between the two graphs differ. WL test is so powerful due to its injective aggregation update that maps different node neighborhoods to different feature vectors.

Multiset -

A multiset is a generalized concept of a set that allows multiple instances for its elements. More formally, a multiset is a 2-tuple X = (S, m) where S is the underlying set of X that is formed from its distinct elements, and $m : S -> N_{\geq 1}$ gives the multiplicity of the elements.
Our framework first represents the set of feature vectors of a given node's neighbors as a multiset, i.e., a set with possibly repeating elements. Then, the neighbor aggregation in GNNs can be thought of as an aggregation function over the multiset. 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. 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.

Deep multisets is the extension of multisets in which we parameterize universal multiset functions with neural networks.

GNNs VS WL Test -

GNNs can be at most as powerful as WL test graph isomorphism test. For a GNN to be as powerful as WL test, it should map two nodes to the same location in the embedding space if and only if the nodes have same identical subtree structures (Multisets) I.e., aggregation scheme must be injective. In WL Test we operate with node labels which cannot capture similarity between subtrees. To distinguish 2 nodes, GNN needs to distinguish structure of their rooted subtrees

$$Power(GNNs) \leq Power(WL)$$

Lemmas and Theorems -

Lemma 1 -

Let $G_1 \; and \; G_2$ be any two non-isomorphic graphs. If a graph neural network A : $G -> \; R^{(d)}$ maps $G_1 \; and \; G_2$ to different embeddings, the WL graph isomorphism test also decides $G_1 \; and \; G_2$ are not isomorphic.

Lemma 2 -

Assume the input feature space X is countable. Let $g^{(k)}$ be the function parameterized by a GNN's k-th layer for k = 1, ..., L, where $g^{(1)}$ is defined on multisets $X \subset \; \chi$ of bounded size. The range of $g^{(k)}$, i.e., the space of node hidden features $h_v^{(k)}$ is also countable $\forall \; k = 1, 2, \ldots, L.$

Lemma 3 -

Assume χ is countable. There exists a function $f : \chi \rightarrow R^n$ so that $h(X) = \Sigma_{x \epsilon X} f(x)$ is unique for each multiset $X \subset \chi$ of bounded size. Moreover, any multiset function g can be decomposed as $g(X) = \phi(\Sigma_{x \epsilon X} f(x))$ for some function ϕ .

Lemma 4 -

There exist finite multisets $X_1 \neq X_2$, so that for any linear mapping W, $\Sigma_{x \epsilon\ X_1} ReLU(Wx) = \Sigma_{x \epsilon\ X_2} ReLU(Wx)$

Theorem 1 -

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 WL Test of isomorphism decides as non-isomorphic, to different embeddings if the following conditions hold:

1. A aggregates and updates node features iteratively with:

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

   where the functions f, which operates on multisets, and ϕ are injective.
2. A's graph-level readout, which operates on the multiset of node features $\{h_v^{(k)}\}$ is injective.

Any GNN that satisfies theorem 3 can learn to map similar graph structures to similar embeddings and capture dependencies between graph structures which will be helpful when the co-occurrence of subtrees is sparse across different graphs or there are noisy edges and node features.

Graph Isomerism Network (GIN)-

Graph Isomerism Network is popular variant in GNNs which has maximum discriminative power in GNNs as this generalizes WL test. GIN satisfies theorem 1. We use theory of deep multisets to model injective multiset functions for the neighbor aggregation.

Injective set functions, such as the mean aggregator, are not injective multiset functions. With the mechanism for modeling universal multiset functions in Lemma 3 as a building block, we can conceive aggregation schemes that can represent universal functions over a node and the multiset of its neighbors, and thus will satisfy the injectiveness condition (1) in the theorem.

Corollary 1 -

Assume χ is countable. There exists a function $f : \chi \rightarrow R^n$ so that for infinitely many choices of ϵ, including all irrational numbers, $h(c, X) = (1 + \epsilon).f(c) + \Sigma_{x \epsilon X} f(x)$ is unique for each pair (c, X), where $c \, \epsilon \, \chi \, and \, X \subset \chi$ is a multiset of bounded size. Moreover, any function g over such pairs can be decomposed as $g(c, X) = \psi((1 + \epsilon).f(c) + \Sigma_{x \epsilon X} f(x))$ for some function ψ.

Corollary (1) provides a simple and concrete formulation among many such aggregation schemes. We can use MLPs to model and learn $f \, and \, \psi$ in the corollary. In practice, we model $f^{(k+1)} \circ \psi^{(k)}$ with one MLP as MLPs can represent the composition of functions. In the first iteration, we do not need MLPs before summation if input features are one-hot encodings as their summation alone is injective. We can make ϵ a learnable parameter or a fixed scalar. Then, GIN updates node representations as -

$h_v^{(k)} = MLP^{(k)}((1 + \epsilon^{(k)}).h_v^{(k-1)} + \Sigma_{u \epsilon N(v)} h_u^{(k-1)})$.

Graph level READOUT function of GIN -

$$h_G = CONCAT(READOUT(h_v^{(k)}|v\epsilon G)|k = 0, 1, \ldots, K)$$

In GIN, the Readout function is sum of all node features in that iteration.

Popular GNN Variants -

1. **1 Layer Perceptron -**

   The function f in Lemma 3 can be parameterized by an MLP but many GNNs use a 1-layer perceptron (like a linear classifier) for graph learning (which makes them not able to distinguish many multisets).

2. **Mean Aggregation -**

   Corollary 2 -

   Assume ψ is countable. There exists a function $f : \psi \rightarrow R^n$ so that $h(X) = \frac{1}{|x|}\Sigma_{x\epsilon X} f(x)$, $h(X_1) = h(X_2)$ if and only if multisets $X_1 \; and \; X_2$ 2 have the same distribution. That is, assuming $|X_2| \geq |X_1|$, we have
   $X_1 = (S, m) \; and \; X_2 = (S, k.m)$ for some $k \epsilon N_{\geq 1}$.

   ```
   If we consider two multisets where the unique
   elements are same but the multiplicity of one
   multiset is k times the multiplicity of the other
   multiset (if a node repeats twice in a multiset it
   repeats 2k times in the other multiset).
   ```

   The mean aggregator maps the both multisets to same embedding as we average over the individual element features i.e., we can say that mean captures the distribution of elements in the multiset but not the exact multiset.

3. **Max Pooling -**

Corollary 3 -

Assume ψ is countable. Then there exists a function $f : \psi \to R^{\infty}$ so that for $h(X) = max_{x \epsilon X} f(x),\ h(X_1) = h(X_2)$ if and only if $X_1\ and\ X_2$ have the same underlying set.

Max-pooling captures neither the exact structure nor the distribution but it identifies the skeleton i.e., the unique nodes of the multiset. So, it is suitable in the cases where the "skeleton" is important rather than distribution like to find the skeleton of a 3D point cloud in which it is robust to noise and outliers.

Datasets used:

We considered Bioinformatic datasets like:

1. MUTAG: -

It has 188(No.of graphs) mutagenic aromatic and heteroaromatic nitro compounds with 7 discrete labels and 2 different classes.

2. PROTEINS: -

It is a dataset where nodes are secondary structure elements (SSEs) and there is an edge between two nodes if they are neighbors in the amino-acid sequence or in 3D space. It has 3 discrete labels, representing helix, sheet or and 2 different classes.

3. NCI1: -

It is a dataset made publicly available by the National Cancer Institute (NCI) and is a subset of balanced datasets of chemical compounds screened for ability to suppress or inhibit the growth of a panel of human tumor cell lines, having 37 discrete labels and 2 different classes.

4. PTC: -

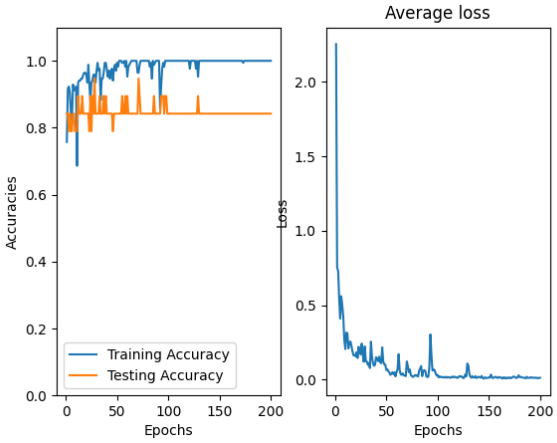
It is a dataset of 344 chemical compounds that reports the carcinogenicity for male and female rats, and it has 19 discrete labels and 2 different classes.

By using these datasets, we are observing which type of aggregation method has better performance.

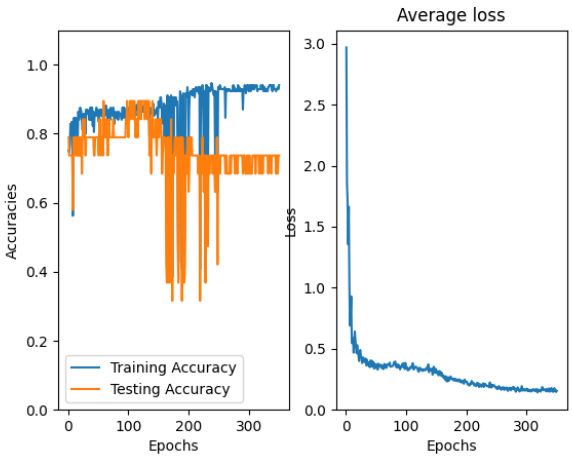
Results

MUTAG: -

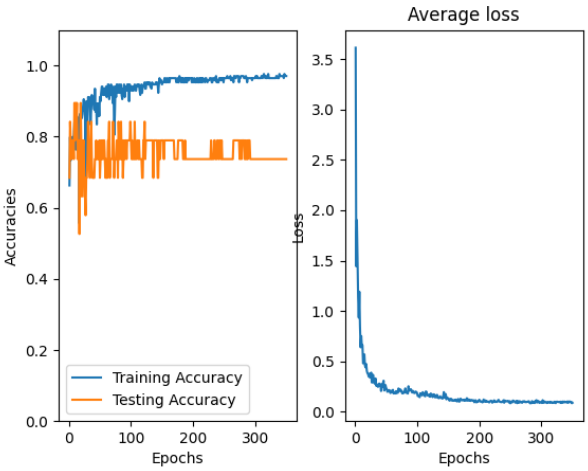
Sum Aggregation: -



Mean Aggregation: -



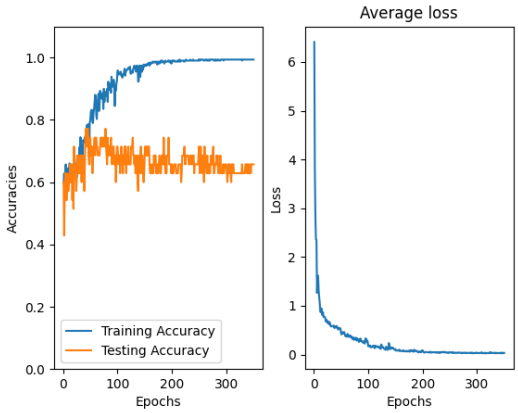
Max Pooling: -



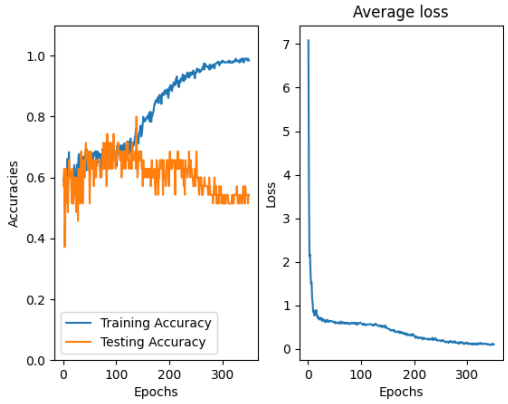
- From the above graphs, we observe that average loss decreases as we perform every epoch, due to the optimization of MLP (For estimation of f and ϕ in Corollary 1).
- We also observed that, sum aggregation led to a faster completion of training and a better testing accuracy rate, when compared with the mean aggregation and max pooling.
- For mean aggregation, we observe many valleys for both training and testing accuracy rate.

PTC: -

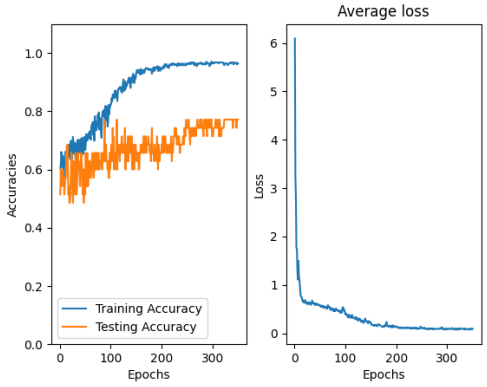
Sum Aggregation: -



Mean Aggregation:



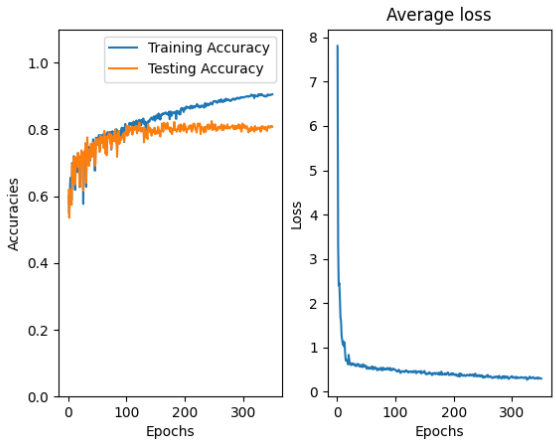
Max Pooling:



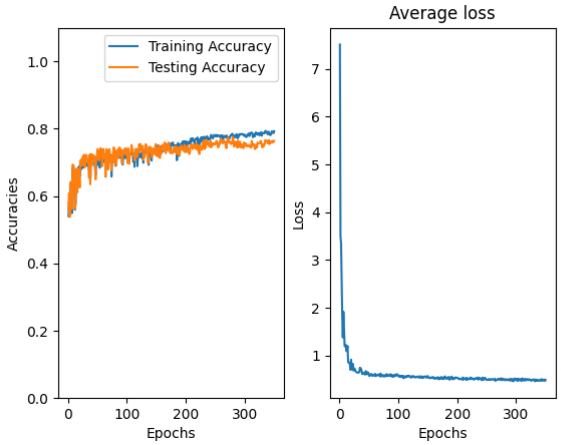
- From the above graphs, we observe that average loss decreases as we perform every epoch, due to the optimization of MLP (For estimation of f and ϕ in Corollary 1).
- We also observed that, sum aggregation led to a faster completion of training a, when compared with the mean aggregation.
- Max pooling has best final testing accuracy compared to Sum aggregation and Mean aggregation.

NCI: -

Sum Aggregation: -



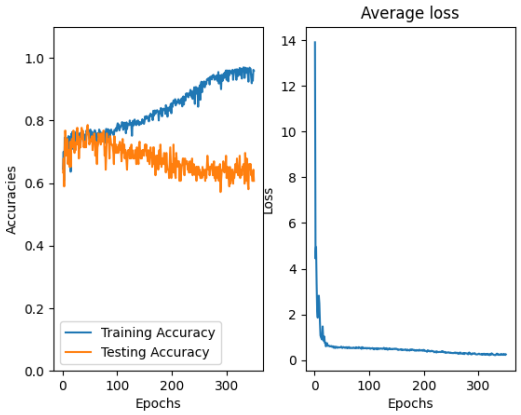
Max Pooling:



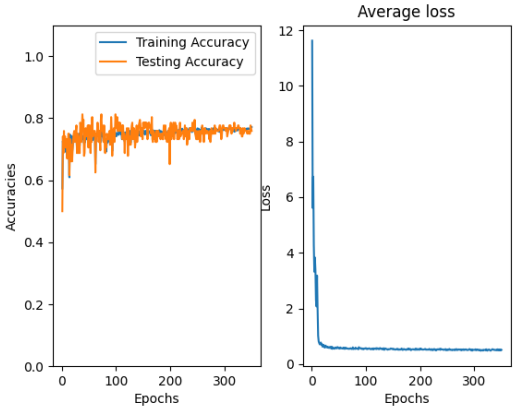
- We observe that, both sum Aggregation and Max pooling doesn't give 100% training accuracy after 350 epochs i.e., Dataset is very difficult to train.
- In Max pooling, Training accuracy and testing accuracy are almost same which may be due to similar skeleton of data.

PROTEINS: -

Sum Aggregation: -



Max Pooling: -



- From the graphs, we observe that Max pooling has better training accuracy rate when compared with sum aggregation.
- In Max pooling, training accuracy is almost constant, not increasing much even with increasing the number of epochs.

Conclusion:

- From above observations, we can conclude that sum aggregation i.e., Graph Isomorphism Network (GIN), we designed has better expressive power compared to other Graph Neural Networks (GCN, GraphSage).

Installation -

pip install scikit-learn matplotlib Pillow

pip install torch

pip install tqdm

pip install numpy

pip install networkx

Test Run -

Open the Codes file

Unzip the dataset file --> unzip dataset.zip

Run the main.py file --> python3 main.py

(this runs the using data set MUTAG which is taken as default)

Rename the output plot file obtained(X.png) to store it.

To run other datasets (for sum pooling) -

python3 main.py --dataset "PROTEINS"

python3 main.py --dataset "PTC"

python3 main.py --dataset "NCI1"

To compare with max/ mean pooling methods -

For a data set selected (in default)

Use commands - python3 main.py --Npooling_method "average" (OR) python3 main.py --Npooling_method "max" (OR) change Npooling_method default to average or max which gives plots respectively.

For multiple changes -

Ex:- python3 main.py --dataset "PROTEINS" --Npooling_method "max" can continue changing required features.

`Link to Github Repo`