Graph Neural Networks

1st Sanket Potdar
Electrical Engineering Department
Indian Institute of Technology, Bombay
Mumbai, India
20d070071@iitb.ac.in

2nd Settipally Nithish Kumar Reddy
Electrical Engineering Department
Indian Institute of Technology, Bombay
Mumbai, India
20d070072@iitb.ac.in

3rd Simran Tanwar
Electrical Engineering Department
Indian Institute of Technology, Bombay
Mumbai, India
20d070078@iitb.ac.in

***Abstract*—Graph Neural Networks(GNNs) are neural networks that use a message-passing approach to update node representations. They are a collection of nodes and the connections between them - edges. This project aims to develop a generative graph neural network model to classify molecules with potential Human immunodeficiency virus infection and acquired immune deficiency syndrome(HIV) inhibition properties. The data-set contains 42,000 molecules represented as SMILES strings and labeled as either inhibitors or non-inhibitors of HIV replication. We developed a generative graph neural network model to accurately classify molecules accordingly by converting them to molecular objects, and extracting node and edge features. Oversampling techniques can help mitigate HIV dataset imbalance and GNN models can be difficult to interpret, making it difficult to develop effective treatments. Selecting appropriate features can be challenging which can be resolved using a featurizer. Adjacency lists or connection matrix can be used to define edges. There is also a challenging problem of smoothening which occurs, if we keep on combining more and more layers making the states indistinguishable from each other, but can be resolved using paranorm. We built a repository to track experiments using MLflow using *dagshub* to make the model's output metrics more accessible and observed the trends which follow the theory.**

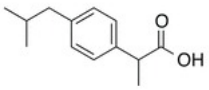
***Index Terms*—Classification, HIV, Graph Neural Network(GNN), node, edge, inhibitors, smiles**

I. Introduction

Graph Neural Networks (GNNs) are a type of neural network that can operate on graphs, which are data structures composed of nodes and edges. GNNs can be used for a variety of tasks, such as node classification, link prediction, and graph classification. At a high level, GNNs use a message-passing approach to update node representations (also known as node embeddings) by aggregating information from their neighbors. The node representations are updated iteratively using a learnable function that takes into account both the node features and the messages received from neighboring nodes. Each node and edge is associated with a feature vector, which is updated through a series of message passing steps. The goal is to learn a compact and informative representation of the entire graph that can be used for downstream tasks, such as node classification, link prediction, and graph-level classification.

The goal of this project is to develop a generative graph neural network model that can classify and generate new molecules with potential HIV inhibition properties. The data-set contains approximately 42,000 molecules represented as SMILES strings and labeled as either inhibitors or non-inhibitors of HIV replication based on their binary label. The SMILES string can be different for a molecule, depending on the notation (a unique molecule can have multiple SMILES strings). Chemical graphs however, are invariant to permutations and so does Graph Neural Networks. So, it is better to represent them as GNN.

We will develop a graph neural network model that can



Ibuprofen
CC(C)Cc1ccc(cc1)C(C)C(0)=0

Fig. 1: Molecule and its SMILES representation(string below in fig)

accurately classify molecules as inhibitors or non-inhibitors. The model will be trained on the labeled data-set and evaluated based on its ability to correctly classify known inhibitors/non-inhibitors. The successful development of a generative graph neural network model can potentially lead to the discovery of new molecules with enhanced HIV inhibition properties and contribute to the development of new antiviral therapies.

HIV is a dataset consisting of classification vased on HIV inhibitor data for 41127 molecules. It is obtained from experimentally measured abilities to inhibit HIV replication. The dataset has been used to train models that estimate the hiv inhibiting ability from chemical structures (as encoded in SMILES strings). HIV dataset is highly imbalanced, meaning that the number of positive examples (inhibitors) is much smaller than the number of negative examples (non-inhibitors). This can make it difficult for the model to learn effectively, as it may tend to predict negative examples more frequently. One way to mitigate this is by using oversampling techniques.

GNN models can be challenging to interpret, which makes it difficult to understand how they arrive at their predictions. This can be a particular challenge in the case of HIV, where understanding the mechanisms of inhibition is critical for developing effective treatments.

Our approach to this is that we started by building a custom

data-set in PyTorch Geometric using the SMILES strings molecules labeled as HIV inhibitors or non-inhibitors. We converted the SMILES strings into node and edge features, which will be used to train a simple graph neural network for classification. After successfully classifying molecules, we can extend the model to generate new molecules with potential HIV inhibition properties using a generative graph neural network. Once we have a functioning model, we build a repository using the Python-based tracking library, MLflow to make the model's output more accessible and graphical.

II. Why GNNs

Graph Neural Networks (GNNs) have gained significant popularity in recent years due to their ability to handle structured data represented as graphs. Compared to traditional neural networks, GNNs can capture and model relationships between entities in a graph, making them suitable for tasks such as node classification, link prediction, and graph classification.

GNNs can be applied to various domains such as social networks, bioinformatics, drug discovery, and recommendation systems. They are particularly useful when the data has an inherent graph structure, and relationships between entities play an important role in the task. For example, in social networks, the relationships between users can be used to predict user behavior, while in bioinformatics, GNNs can be used to predict protein interactions or chemical properties.

In summary, one should choose GNN for a machine learning project when dealing with data that can be represented as a graph, and when relationships between entities in the graph are important for the task at hand.

III. Dataset preprocessing

Dataset preprocessing is a crucial step in Graph Neural Networks (GNNs) that involves transforming raw graph data into a format that can be efficiently processed by the GNN model. In this, we read and gather what all information is given in the data. Graph features such as node features and edge features need to be extracted from the data. To achieve this, predefined functions such as GetAtomicNum(), GetDegree(), GetFormalCharge(), and GetBond() can be used to extract node and edge features for molecules. Additionally, adjacency matrices can be defined using a set of functions. In graph theory, adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph. In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix with zeros on its diagonal. We can also use deepchem featurizers to extract features more easily for PyTorch Geometric. These featurizers provide several features that can be used to easily construct the dataset. Overall, these preprocessing steps are essential in preparing the data for GNNs, allowing for more efficient and effective graph-based predictions.

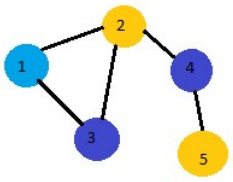


Fig. 2: Simple graph

	v1	v2	v3	v4	v5
v1	0	1	1	0	0
v2	1	0	1	1	0
v3	1	1	0	0	0
v4	0	1	0	0	1
v5	0	0	0	1	0

Fig. 3: Adjacency Matrix for above graph

IV. Graph level prediction

Graph-level prediction refers to making predictions or classifications based on entire graphs, rather than on individual nodes or edges within the graphs. In other words, the goal is to predict a label or property for the entire graph, such as predicting the functional properties of a molecule or the category of a citation network. It is a common task in graph-based machine learning, where the input is represented as a graph and the output is a single label or value associated with the entire graph. This is in contrast to node-level or edge-level prediction, which involves making predictions for each individual node or edge in the graph. Graph-level prediction can be performed using a variety of machine learning models, including Graph Neural Networks (GNNs), Support Vector Machines (SVMs), Random Forests, and Convolutional Neural Networks (CNNs). These models can be trained on graph datasets and can learn to extract features from the graphs to make accurate predictions about the graph as a whole.

A. Naive global pooling

Naive Global Pooling is a simple method of summarizing the node-level features of a graph to obtain a graph-level representation. It involves taking the average, maximum, or sum of the node features across all nodes in the graph to obtain a single vector that represents the entire graph.

Naive Global Pooling is a straightforward method that does not take into account the structure or relationships between the nodes in the graph. As a result, it may not capture the important characteristics of the graph, and may result in a loss of information. However, it can be a useful baseline approach for graph-level prediction tasks, and can be combined with other more sophisticated methods such as Graph Convolutional Networks (GCNs) or Graph Attention Networks (GATs) to improve performance.

B. Heirarchical pooling

Hierarchical pooling is a technique used in Graph Neural Networks (GNNs) to reduce the size of a graph by iteratively removing nodes and aggregating information. The idea behind

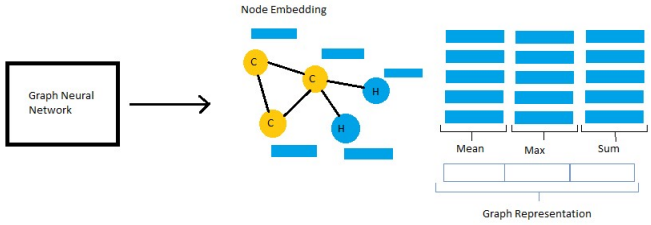


Fig. 4: Naive Global Pooling

hierarchical pooling is to capture the important features of a graph at multiple levels of granularity, from the node level to the graph level. It is to use techniques similar to Convolutional Neural Networks (CNNs), where pooling operations are performed to reduce the dimensionality of the input data.

In GNNs, pooling can be used to combine the features of adjacent nodes or subgraphs into a single representation, which can then be used to update the features of the next level of nodes or subgraphs. Nodes can be reduced through techniques such as differentiable pooling, which clusters similar nodes together, or top-K pooling, which simply drops the least important nodes based on a learn-able vector.

To capture information from multiple levels of granularity, intermediate results from pooling operations can be aggregated and added to the final graph-level representation. This can be achieved through concatenation or other aggregation methods. Hierarchical pooling is a powerful technique that can be used to reduce the size of a graph while preserving important features and information. It can be combined with other techniques such as GCNs and GATs to achieve state-of-the-art performance on graph-based machine learning tasks.

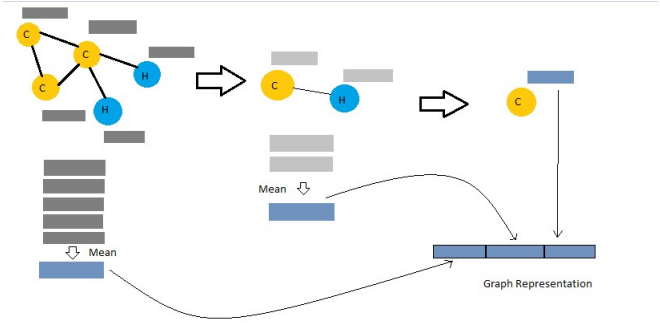


Fig. 5: Heirarchical Pooling

C. Super node

Super nodes refer to a technique used in Graph Neural Networks (GNNs) to combine multiple nodes in a graph into a single node, or ”super node”, in order to reduce the size of the graph and improve computation efficiency. This can be useful in situations where a graph contains a large number of nodes and edges, making it computationally expensive to perform node-level computations.They can be created by clustering nodes together based on their similarity, or by selecting a subset of nodes that are representative of the graph as a whole. The features of the super node are then computed by aggregating the features of the constituent nodes.

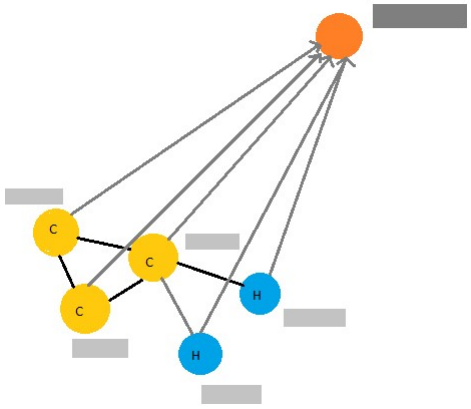


Fig. 6: Super Node

V. Approach

A. Oversampling

We oversampled the data as we hace aprroximate 40:1 ratio for inhibitors and non-inhibitors. So, we loaded the raw dataset from the CSV file and applies oversampling to balance the classes of the target variable, ”HIV-active”.The oversampling is done by replicating the positive class (where ”HIV-active” is 1) until the number of positive samples is the same as the number of negative samples. The resulting dataset is then shuffled and saved as a new CSV file. This new CSV file contains an additional ”index” column that serves as an ID for each sample in the dataset.

B. Custom data-set in PyTorch Geometric

We started by building a custom data-set in PyTorch Geometric using the SMILES strings molecules labeled as HIV inhibitors or non-inhibitors.
For custom data-set, we gathered information given in the data and extracted node features, edge features and adjacency matrix for each molecule. The dataset class inherits from PyG’s Dataset class and overrides the required functions ”raw-file-names”, ”processed-file-names”, and process to define the way the dataset is downloaded, saved, and preprocessed. In particular, the ”MoleculeDataset” class defines a ”process” method to pre-process molecular data, including extracting node and edge features, adjacency information, and labels from SMILES strings. The preprocessed data is then saved as PyG Data objects in the processed directory.

C. Graph Classification

We then developed a graph neural network model that can accurately classify molecules as inhibitors or non-inhibitors. The GNN model takes a graph represented as a set of nodes and edges as input, where each node has a set of features (represented by a feature vector) and each edge has a set of

attributes. The goal of the model is to predict a scalar value for each input graph. The GNN model architecture consists of several layers. The first layer is a TransformerConv layer that transforms the input node features and edge attributes into a new feature space using multi-head attention. The resulting feature vectors are then passed through a linear layer, batch normalization, and ReLU activation. This process is called a transformation layer, and its purpose is to learn a new representation of the input graph that is better suited for downstream processing.

The remaining layers in the model are identical and are called convolutional layers. These layers apply the same transformation to the input graph as the first layer, except that they take the output of the previous layer as input. After each convolutional layer, the output is passed through a linear layer, batch normalization, and ReLU activation.

To reduce the size of the graph, a 'TopKPooling' layer is applied after every "self.top-k-every-n" convolutional layers. This layer reduces the number of nodes in the graph by keeping only the top k nodes based on a learned score function. The remaining nodes and their corresponding edges are used as input to the next convolutional layer.

D. Training data

First split the data into training and testing and then The "train-one-epoch" function trains the model for one epoch, using the "train-loader" DataLoader to load batches of data, and the "optimizer" and "loss-fn" to optimize the model's parameters. The test function evaluates the model on the test set, using the "test-loader" DataLoader and the same "loss-fn". Both functions calculate several evaluation metrics such as F1 score, accuracy, precision, and recall, and log them to MLflow. The "log-conf-matrix" function generates a confusion matrix as a heatmap using the Seaborn library, and logs it to MLflow as an artifact. The "calculate-metrics" function calculates the evaluation metrics and logs them to MLflow.

The hyperparameters and best parameters are defined for the machine learning model, along with the input and output schema for the model. The input schema defines the shape and data type of the model's input, while the output schema defines the shape and data type of the model's output.There is also a challenging problem of smoothening which occurs, if we keep on combining more and more layers making the states indistinguishable from each other, but can be resolved using paranorm. This code uses the MLflow library to create a signature for the machine learning model.

This code define a function named run-one-training(params) that trains a graph neural network (GNN) model on a molecular dataset for binary classification. The function uses "mango" for hyperparameter tuning, "mlflow" for experiment tracking, and PyTorch for model training.The function first logs the hyperparameters used for this experiment, then loads the molecular dataset, and sets up the training and testing data loaders. It then loads the GNN model with the specified hyperparameters, and sets up the loss function, optimizer, and learning rate scheduler.

The training loop runs for 80 epochs, during which the model is trained on the training data, and the loss is logged. Every 5 epochs, the model is evaluated on the testing data, and the loss is logged. If the current testing loss is better than the previous best loss, the model is saved, and the early stopping counter is reset. If the early stopping counter reaches 10 (indicating no improvement for 50 epochs), the function stops and returns the best testing loss. The function returns a list containing the best testing loss.

VI. Experiments and Results

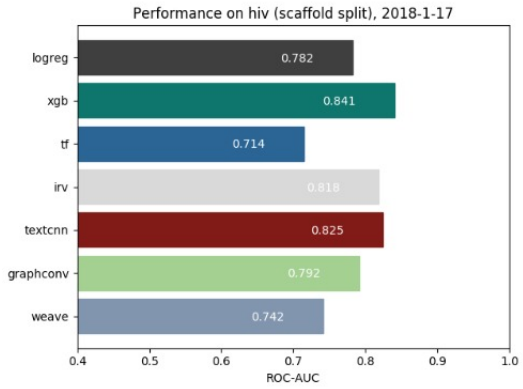


Fig. 7: Performance of dataset on different models [12]

After hyperparameter tuning, we get the accuracy to be 72%. The trends for the performance metrics and train,test loss have been observed using mlflow.

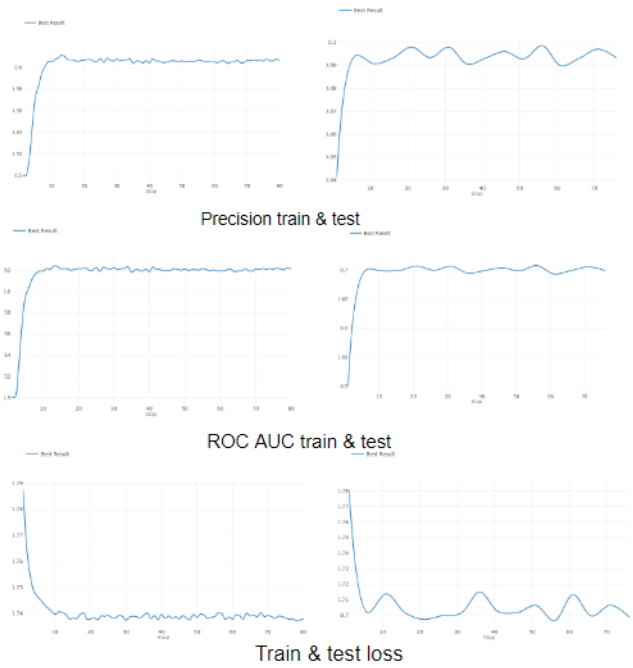


Fig. 8: Metric Trends

Name	Value
batch_size	64
learning_rate	0.001
model_attention_heads	1
model_dense_neurons	128
model_dropout_rate	0.1
model_embedding_size	8
model_layers	6
model_top_k_every_n	2
model_top_k_ratio	0.4

Fig. 9: Best parameters

VII. Discussion and Conclusions

We could sucessfully classify the molecules into HIV inhibitors correctly. We used hyperparameters like weight decay, batch size, learning rate , layers, etc. and optimized or tuned them to find the best hyperparameters using Mango. We created our custom dataset by appending node features, edge features, adjacency matrix and labels. We created a model using TokKPooling, Graph transformer convolution layers and linear transformer layers with batch normalization and trained and tested our custom dataset on it. We used the metrics - precision, accuracy, recall, f1-score, RUC-AUC to analyze our result using tracking software - MLflow by creating an experiment respository on Dagshub. We could observe the suitable trends where precision and RUC-AUC increases till a value and reaches a saturation. Also, the RUC-AUC value for both train and test data is above 0.5 for best hyperparameters which signifies a good classifier. The loss also follow the theoretical trend of decreasing and saturating to a value. We observe the accuracy on test set to be around 74% . We trained it till 80 epochs.

VIII. Future scope

Hyperparameters can be further tuned to get improved performance.

The practicality of this project is huge. Here, we classified the molecule in to classes HIV inhibitor or not.Long term goal of this project is to generate HIV inhibitor molecules. Variational auto encoders(VAE) is one of the popular approach.In a nutshell, a VAE is an autoencoder whose encodings distribution is regularised during the training in order to ensure that its latent space has good properties allowing us to generate some new data. Moreover, the term “variational” comes from the close relation there is between the regularisation and the variational inference method in statistics. Such generative processing can generate HIV inhibitor molecules and can be used to discover new drugs and have tremendous medical applications.

IX. Acknowledgment

It is a great pleasure to express our deepest gratitude to **Professor AMIT SETHI** for giving us this life changing experience and prompt guidance throughout the course.

X. Key Links

- HIV Dataset
- Colab File
- Github Repository
- MLflow results obtained on colab
- Video Explanation

References

[1] Thomas N. Kipf, Max Welling, Semi-Supervised Classification with Graph Convolutional Networks, Submitted on 9 Sep 2016 (v1), last revised 22 Feb 2017 (this version, v4), https://arxiv.org/abs/1609.02907v4
[2] Random Oversampling and Undersampling for Imbalanced Classification by Jason Brownlee
[3] rdkit docs
[4] Run rdkit and deep learning on Google Colab! RDKit
[5] open source rdkit application colab
[6] open source gnn training using pytorch
[7] Github-Deep chem
[8] MolGraphConvFeaturizer deepchem docs
[9] pytorch docs
[10] MLflow — With Colab published by point collections
[11] ARm-mango for hyperparameter scheduler and tuner
[12] Results and Performances of HIV dataset