WARSAW UNIVERSITY OF TECHNOLOGY

MACHINE LEARNING ON GRAPHS

PROJECT

by

Abba Umar, and Swetha Suresh Kumar

Table of Contents

1. Abstract

This report presents a graph machine learning pipeline for HIV molecule prediction using Graph Neural Networks (GNNs) and PyTorch Geometric. A custom graph dataset is created by transforming molecular representations into graph structures. The GNN model is trained to classify molecules as suitable or unsuitable for HIV inhibition. The pipeline involves data pre-processing, GNN model construction, and training with various techniques for improved performance. Additionally, a generative model is developed to generate new molecules with potential HIV inhibitory properties.

2. Task description

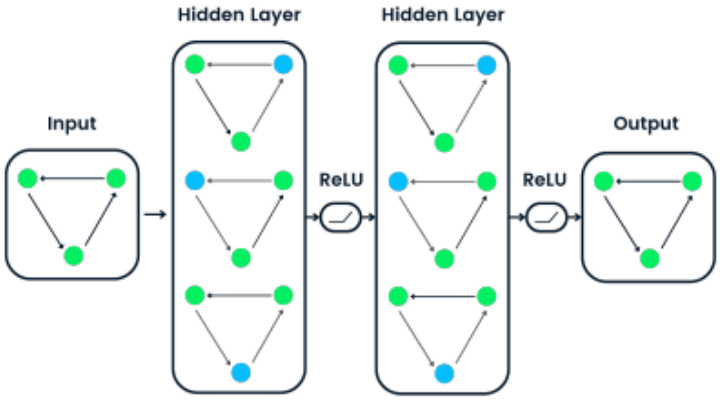
The objective of this task is to prepare a full graph machine learning pipeline with GNN models applied to non-graph data. Specifically:

2.1. propose a non-trivial/creative way to infer graph structure from your data. Distance graphs and k-nearest neighbours' graphs are usually not particularly creative, but that depends on the specific case.

2.2. design and train a GNN of your choice, with layers of possibly various types and shapes,

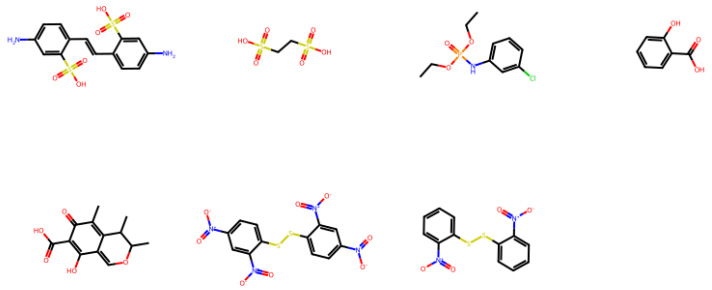
2.3. compare the performance of your GNN with a standard non-graph-based neural network model, e.g., CNN, RNN, LSTM, etc.

3. Introduction



Graph-structured data is abundant in real-world applications, ranging from biology networks to social networks and molecular graphs. Graph neural networks (GNNs) have emerged as powerful models for learning representations from graph data. GNNs have shown promising performance and undergone significant methodological advancements, making them a popular choice for capturing complex relationships and dependencies in graph-structured data.

The dataset used in this project consists of 41,127 molecules represented by SMILES strings. SMILES (Simplified Molecular Input Line Entry System) is a textual representation of chemical structures that allows for easy storage, sharing, and analysis. There are 39,684 molecules that do not possess inhibitory properties against HIV and 1,443 molecules that have the potential to inhibit HIV replication. This indicates that the dataset is imbalanced, with a majority of molecules being non-HIV-active and a smaller portion being HIV-active.



4. ML Task Overview

The ML task at hand is binary classification of molecules into HIV active or inactive. The dataset consists of molecular structures represented as SMILES strings. The baseline model used is a graph neural network (GNN) implemented using the PyTorch Geometric library. The GNN model takes into account the graph structure of molecules and learns to extract relevant features for classification.

The task involves the following technical steps:

4.1. Dataset pre-processing: The molecular data is transformed into graph structures, with node and edge features

Node Features: Atomic Number, Atom Degree, Formal Charge, Hybridization, Aromaticity

Edge Features: Bond Types, Ring Information

4.2. GNN model architecture: The GNN model incorporates the following modules and layers, each with its respective formula:

4.2.1. GATConv (Graph Attention Convolution) module:

1. Node Representation Update: $NR_i = \sigma(\sum_{j \epsilon N(i)} \alpha_{ij} \times Linear(NR_j))$
2. Attention Coefficients: $\alpha_{ij} = \frac{\exp\left(LeakyReLU(a^{-T}[NR_i || NR_j])\right)}{\sum_{k \epsilon N_i} \exp\left(LeakyReLU(a^{-T}[NR_i || NR_k])\right)}$

4.2.2. TopKPooling layer: The TopKPooling layer performs graph pooling, which reduces the number of nodes in each graph by selecting the top-k nodes based on their importance scores.

4.2.3. Linear layer: The linear transformations are implemented using the ***torch.nn.Linear*** class in the code. It applies a linear transformation to the node representations, projecting them into the desired output dimensions.

4.3. Model training: The GNN model is trained using the training dataset, involving the following steps:

4.3.1. Forward Pass:

- The input features (**x**), edge attributes (**edge_attr**), edge indices (**edge_index**), and batch indices (**batch_index**) are passed through the GNN model.
- The model applies graph convolutional layers (**GATConv**), pooling layers (**TopKPooling**), and linear layers (**Linear**) to transform the input features and capture graph-level representations.

4.3.2. Loss Computation:

- The predicted labels (**pred**) obtained from the model are compared with the true labels (**y**) using the loss function (**CrossEntropyLoss**).

- The loss function computes the cross-entropy loss between the predicted probabilities and the true labels.

4.3.3.Backward Pass (Backpropagation):

- The gradients of the loss with respect to the model parameters are computed.
- These gradients are propagated back through the model using the chain rule of derivatives.
- The optimizer (**SGD**) uses these gradients to update the model parameters and minimize the loss.
- The learning rate determines the step size of the parameter updates.

4.3.4.Learning Rate Scheduling:

- A learning rate scheduler (**ExponentialLR**) is used to adjust the learning rate over epochs.
- In the project, an exponential decay schedule with a decay factor (**gamma**) of 0.95 is used.
- This scheduler gradually reduces the learning rate, allowing the model to converge more effectively.

5. Graph inference method

The graph inference method is performed using the trained GNN model to make predictions on new, unseen graph data.

5.1. Data Preparation: First, the test dataset is loaded using the **test_loader** data loader, which provides batches of graph data for inference.

5.2. Model Evaluation: The code iterates over each batch of the test dataset using a **for** loop. For each batch, the model is put into evaluation mode using **model.eval()**.

5.3. Forward Pass: The GNN model is applied to the input graph data using the **forward** method. The input graph data includes the node features (**batch.x.float()**), edge attributes (**batch.edge_attr.float()**), edge indices (**batch.edge_index**), and batch indices (**batch.batch**). These inputs are passed to the GNN model to obtain predicted outputs.

5.4. Prediction: The output of the GNN model is stored in the **pred** variable, representing the predicted labels or logits for each input sample in the batch.

5.5. Metrics Calculation: The predicted outputs (**pred**) and ground truth labels (**batch.y)** are converted from tensors to numpy arrays (**cpu().detach().numpy()**) to perform further calculations. Various evaluation metrics such as F1 score, accuracy, precision, and recall are then calculated based on the predicted and true labels using functions like **f1_score(), accuracy_score(), precision_score(), and recall_score().**

6. Implementation details and challenges

The project uses the following technologies/libraries: pandas, rdkit, torch_geometric, numpy, os, and torch.

6.1. The implementation consists of the following steps:

1. Data Preprocessing:
    - The code defines a custom dataset class, **HIVDataset**, which extends the **torch_geometric.data.Dataset** class.
    - The dataset class processes the input data by converting the SMILES representation of molecules into RDKit molecule objects.
    - It extracts node features (atomic properties) and edge features (bond properties) from the RDKit molecules.

- The adjacency matrix (edge indexes) is constructed based on the molecule's connectivity.
- The binary label indicating the molecule's HIV activity is retrieved.
- The extracted features and labels are saved as **Data** objects for each molecule.

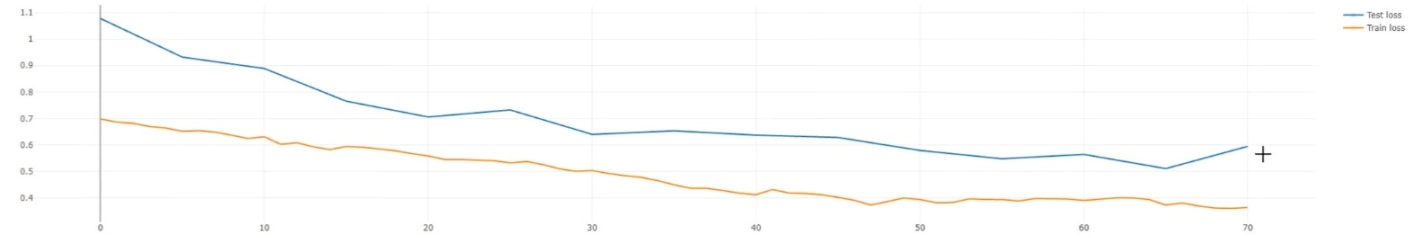
2. Model Architecture:
    - The code defines a GNN model class, **GNN**, which inherits from **torch.nn.Module**.
    - The model architecture includes multiple layers of graph convolution (GATConv), graph pooling (TopKPooling), and linear transformation (Linear) operations.
    - The GNN model takes node features, edge features, edge indexes, and batch indexes as input.
    - The forward pass applies the graph convolution, pooling, and linear transformations to generate the final prediction.
3. Training and Evaluation:
    - The code initializes the model, optimizer (SGD), scheduler (ExponentialLR), and loss function (CrossEntropyLoss).
    - The **train()** function performs the training loop using mini-batches from the training dataset.
    - In each training iteration, the model is trained on a batch of graph data, and the loss is calculated and backpropagated to update the model's parameters.
    - The **Predict()** function performs the evaluation loop using mini-batches from the test dataset.
    - After each epoch, the model's performance is evaluated using metrics such as F1 score, accuracy, precision, and recall.
    - The learning rate scheduler is updated to adjust the learning rate during training.
    - The training and evaluation processes are repeated for a specified number of epochs.

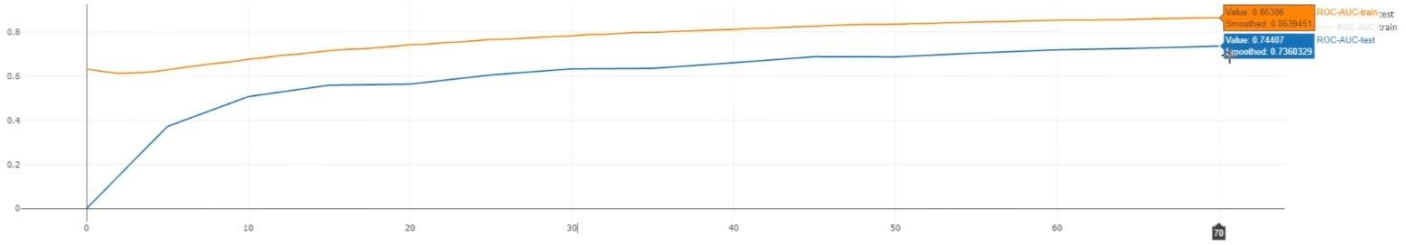
6.2. Challenges

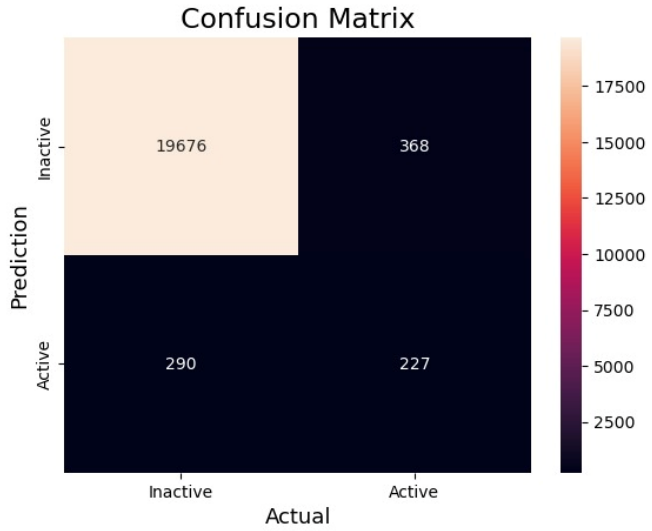
- Graph Data Representation: One of the main challenges encountered was representing the molecular structure as a graph. The complex connectivity between atoms and bonds required careful consideration to accurately capture the structural information in a graph format.
- GNN Architecture and Components: Understanding and implementing the Graph Neural Network (GNN) architecture using libraries like torch_geometric posed its own challenges. Familiarizing oneself with various components, such as GATConv (Graph Attention Convolution) and TopKPooling, and effectively incorporating them into the model required a thorough understanding of their functionality and usage.
- Edge Attribute Support: One specific challenge arose from the lack of support for edge attributes in the current PyTorch implementation. Edge attributes play a crucial role in capturing additional information and relationships between nodes in a graph. Finding a workaround or alternative solution to incorporate edge attributes into the GNN model became a significant hurdle.

7. Results

The results obtained:







8. Observation

- The dataset used for training the model contains 41,127 samples, with 39,684 samples labelled as 0 (HIV inactive) and 1,443 samples labelled as 1 (HIV active).
- The model architecture used is a Graph Neural Network (GNN) with three graph convolutional layers and pooling layers, followed by fully connected layers.
- The model was trained for 500 epochs, and the training loss decreased over time.
- The evaluation metrics on the training data after the last epoch are as follows:
    - F1 Score: 0.790
    - Accuracy: 0.967
    - Precision: 0.972
    - Recall: 0.710
- The confusion matrix shows the following counts:
    - True Negative (TN): 19676

- False Positive (FP): 368
- False Negative (FN): 290
- True Positive (TP): 227

9. Conclusion

- The model achieved moderate performance on the training data, with an F1 score of 0.790 and an accuracy of 0.967.
- The model correctly predicted a large number of instances as negative (true negative) and a smaller number of instances as positive (true positive). However, it made some false positive predictions and false negative predictions.
- The precision is slightly higher than the recall, indicating that the model is better at correctly predicting the HIV inactive class (0) than the HIV active class (1).
- The performance of the model could be further improved by tuning hyperparameters, increasing the size of the training data, or using more advanced architectures or techniques.

10. Reference

- MoleculeNet (2022), "Hiv." URL https://moleculenet.org/datasets-1.
- Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2017). Graph Attention Networks. ArXiv. /abs/1710.10903
- Haoyang Li, Xin Wang (2022), "Ood-gnn: Out-of-distribution generalized graph neural network." URL http://mn.cs.tsinghua.edu.cn/xinwang/PDF/papers/2022_Out-of-Distribution%20Generalized%20Graph%20Neural%20Network.pdf
- AbidAliAwan (2022), "Graph neural networks." URL https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial