Molecule classification with a GNN

Jonas Quenzer, Patrikc Pollek
Department of Computer Science, University of Vienna, Austria

***Abstract*—**

I. Introduction

Graphs are deeply embedded in our day-to-day experiences and intellectual pursuits. For instance, consider molecules as a form of graphs, with each node symbolizing an atom and each edge representing a chemical bond. Similarly, social networks can be perceived as graphs too, where each node might denote a person, and an edge could signify various forms of relationships. These are only two examples of a wide range of applications of this complex data structure. In practical scenarios, we often encounter problems where we need to make predictions using specific graphical data. For instance, we might need to categorize molecules into two groups, based on their ability to inhibit the replication of the HIV virus. Given the significant success of Deep Neural Networks, it seems sensible to explore the possibilities of merging these two fields together for more effective solutions.

II. Problem setting

In this work, we classify molecule graphs into two classes based on whether or not they inhibit the replication of the HIV virus. The Dataset was taken from [1] and contains 41,127 graphs where each node has a 9-dimensional feature vector that contains information about the bond type or chirality for example. The graphs have varying numbers of nodes and edges but on average 25 nodes and 27 edges. The dataset is unbalanced which means we have much more molecules that do not inhibit the replication. The evaluation metric for this task is the ROC AUC score which is the Area Under the Receiver Operating Characteristic Curve that plots the true positive rate (TPR) against the false positive rate (FPR). The True Positive Rate (TPR), also known as Sensitivity or Recall, is calculated as:

$$TPR = \frac{TP}{TP + FN} \tag{1}$$

where TP is the number of true positives and FN is the number of false negatives.

The False Positive Rate (FPR), also known as Fall-Out, is calculated as:

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

where FP is the number of false positives and TN is the number of true negatives.

III. Naive Approach

Typically a graph is represented by its adjacency matrix A on a computer. A first naive approach would be to feed a flattened version of A into a feed-forward neural network and predict the outcome. But this idea will lead to immediate problems. The first problem is that each graph has a different amount of nodes which results in different sizes of the matrix A and the fact that we are not using the node's feature vectors at all but they might have relevant information. This first part would require us to introduce some kind of padding to have a fixed input size to our network. The second problem is a bigger one namely that we have no intrinsic ordering of our nodes. A permutation of rows of the matrix A results in the same graph/molecule but obviously this destroys the learned relationship between weights and input. We call this an order-sensitive model. To solve this issue we want to create a model which is permutation invariant meaning for a matrix A and a permutation matrix P the following holds :

$$f(PAP^T) = f(A)$$

IV. Message Passing and GNN

The key feature of a Graph Neural Network is the message-passing framework. The goal is to use the node feature vector x_u of a node u to learn an embedding z_u. Message passing is an iterative approach where in each iteration we update the (hidden) node embedding h_u^k of node u according to some information from the neighborhood of the node $N(u)$. This can look like this :

$$h_u^{k+1} = UPDATE^k(h_u^k, AGGREGATE^k(\{h_v^k, \forall v \in N(u)\}))$$

After K rounds we define the final embedding for each node as $z_u = h_u^K \forall u$. Taking a permutation invariant function as AGGREGATE such as average, sum, and so on makes the Network permutation invariant by design.

V. GCN

Taking different AGGREGATION and UPDATE functions is the most important distinction between different variants of GNN. A very powerful instance is the graph convolutional network where we weigh the embedding of each neighbor based on the degree of the neighbor. We also use self-loops ($A_{ii} = 1$) to omit the explicit update step and also take the embedding of the node itself into consideration when aggregating. This results in the following update :

$$h_u^k = \sigma(W^k \sum_{v \in \{N(u), u\}} \frac{h_v}{deg(v)})$$

where W is a weight matrix learned by the network. So the AGGREGATION is a weighted sum followed by a nonlinear function σ called the activation function. This single update can be called a graph convolutional laver and a series of these updates is called a graph convolutional network.

Applying these interactions will result in an embedding of the nodes. To get an embedding of a whole graph z_G which then can help us classify each graph will need one more step called graph pooling.

We can again add up the embeddings of all nodes to end up with a so-called readout layer (In this specific case with the global sum pooling layer). This would result in

$$z_G = \frac{\sum_v z_u}{|V|}$$

, where $|V|$ is the number of all nodes. Again we could also take the average or max for example.

From this, we can then use standard NN layers to predict the class.

A. Other Layers

VI. Method

To build a molecule classifier we tried several different variations of GNN. These include GCN, transformer layer etc.

We used the provided data loader from [1] to load the data and also split the data according to the prearranged splits. We then build several GNNs with the PyTorch geometric library. We used the CrossEntropyLoss to evaluate the prediction and used Adam with varying learning rates to train the model. We used a validation set to choose the best architecture for each variant and then finally after training evaluated it on the test set. The results can be see in table **??**.

VII. Results

VIII. Summary

References

[1] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.