A Comparision between Graph neural network models including:

Graph Convolutional Neural Network, Graph Attention Network (GAT), and GraphSAGE

for node classification of Graphs obtained from

Protein-Protein Interaction and Gene-Disease Assosiation

1. Introduction

In this work, the task is to do binary classification on proteins (the term gene is also used interchangeably) if it is related to a specific disease or not and compare the results of three different models (GCN, GAT. The assumption is that genes associated with the same or similar diseases tend to accumulate in the same neighborhood of the molecular network. Therefore, a key step is to measure the distance between candidate genes and known disease genes in the protein-protein interaction (PPI) network.
Note: all results of this work is not done on a reduced version of PPI but on the actual version

1. Data preparation:

I. Before going deep into models we need to prepare our data by the use of two files provided. The first one is the Protein-protein interaction file and the other one is a file with information about different diseases and more importantly the gene causing the disease. For example, in this file which is called $gene\_disease\_association$, there are different diseases which in this work I chose two different diseases to compare the results: Asthema and Diabetes. In this table, only disease name and geneSymbol columns are important for me and by some lines of codes, I create a new lighter data frame to be used later.

II. On the other hand, in the PPI file each line shows the interaction between genes/protein. At first glance, it is obvious that this file can be also represented as a graph where each node is a gene and simply there is an edge between two nodes if they can be seen in the same line in the PPI file. I will create the graph G by the use of the $networkx$ library

III. Another step of our data preparation(which is done alongside step 2) is to give values/labels to each protein. in other words, for each node in the graph obtained from PPI, I check if it is causing diabetes/Neoplasms By comparing the node with nodeSymbol in df obtained in step1. If so, I associate the label True/1 to it and false otherwise.

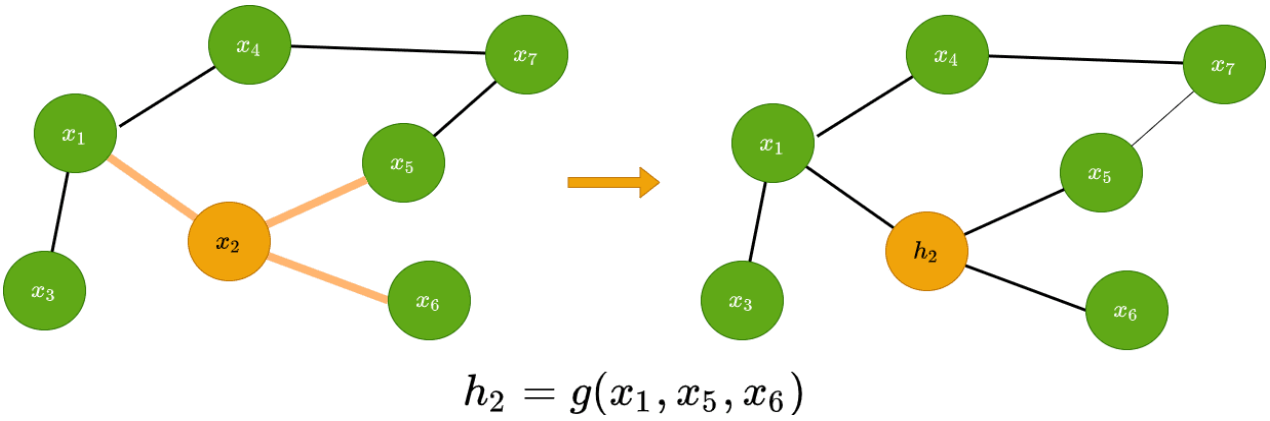
IV. I need to consider unbalanced Graphs. In this study, there are fewer genes causing disease (1) among all genes(0,1). Hence, I define an unbalanced graph if proportion of genes not causing disease in the graph is too high so I need to under-sample them with function and continue my work on that graph

Now our data is ready to be fed into our machine learning models. in this work I compare results and different metrics of these two diseases with and without balancing the graph. But before that lets have a glimps on what are essence of these models.

2. Models:

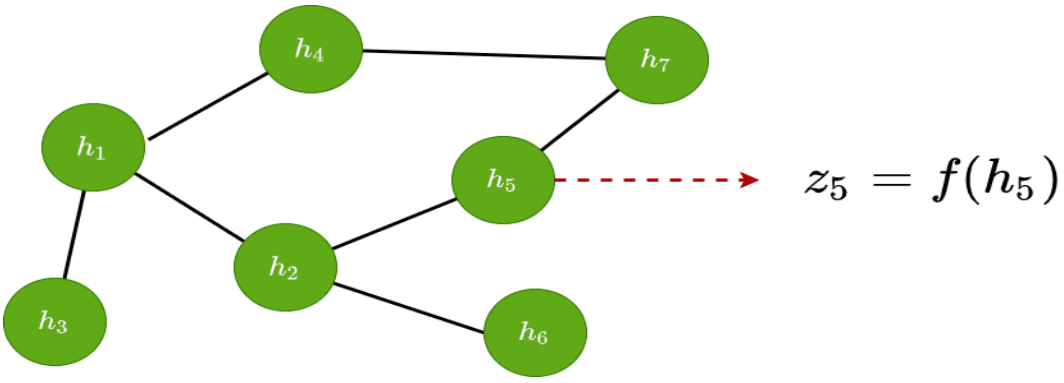
Graph Neural Networks (GNNs) are an effort to apply deep learning techniques in graphs. The term GNN is typically referred to a variety of different algorithms and not a single architecture. From now on, a node's feature vector will be denoted as hi, where *i* is the node's index. Similarly an edge's feature vector will be denoted as *eij*, where *jj* are the nodes that the edge is attached to(I will not discuss edge features, edge classification, etc). The basic idea behind most GNN architectures is **graph convolution.** In essence, we try to generalize the idea of convolution into graphs. Graphs can be seen as a generalization of images where every node corresponds to a pixel connected to 8 (or 4) adjacent neighbours. Graph convolution predicts the features of the node in the next layer as a function of the neighbours' features. It transforms the node's features *xi*

in a latent space *hi* that can be used for a variety of reasons. Visually this can be represented as follows:



$$h_2 = g(x_1, x_5, x_6)$$

If we apply a shared function *f* to each of the latent vectors *hi*, we can make predictions for each of the nodes. That way we can classify nodes based on their features: $Zi=f(hi)$



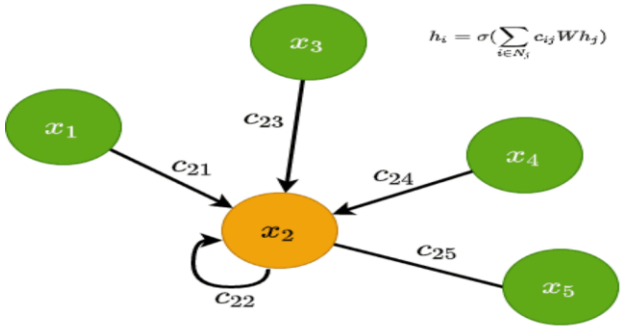
2.1. GCN:

if *H* is the feature matrix and *W* the trainable weight matrix, the update rule for the GCN layer becomes $H(l+1)=\sigma(H(l))$

From a node-wise perspective, the update rule can be written as :

$$h_i^{(l)} = \sigma(\sum_{i \in N_j} c_{ij} W h_j)$$

Where $c_{ij} = \frac{1}{\sqrt{|N_i||N_j|}}$, and N_i and N_j are the sizes of the nodes' neighbourhoods.



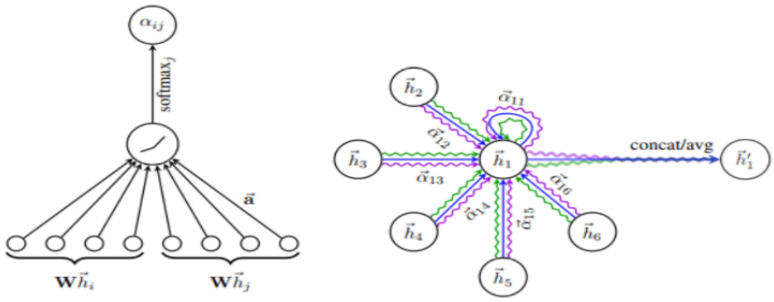
2.2. GAT:

The coefficient cij in GCN is derived from the degree matrix of the graph and is heavily dependent on the structure of the graph. Intuitively, it represents how important the node's *j* features are for node *i*. The main idea behind GAT is to compute that coefficient implicitly rather than explicitly as GCNs do. That way we can use more information besides the graph structure to determine each node's "importance", by considering the coefficient to be a learnable attention mechanism. the coefficient, from now on denoted as *aij*, should be computed based on node features, which are then passed into an attention function. Note that edge features can also be included. Finally, the Softmax function is applied in the attention weights *ai* to that result in a probability distribution. Mathematically we have:

$$a_{ij} = attention(h_i, h_j)$$

$$a_{ij} = \frac{exp(a_{ij})}{\sum_{k \in N_i} exp(a_{ik})}$$

Visually this can be seen on the left side of the following image

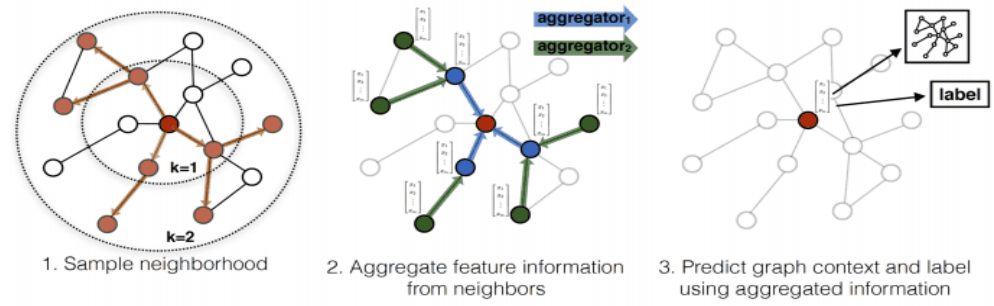


2.3. GraphSAGE:

One major drawback of most GNN architectures is scalability. In general, each node's feature vector depends on its entire neighborhood. This can be quite inefficient for huge graphs with big neighborhoods. To solve this issue, sampling modules have been incorporated. The main idea of sampling modules is that instead of using all neighborhood information, we can sample a subset

of them to conduct propagation. GraphSage popularized this idea by proposing the following framework:

1. Sample uniformly a set of nodes from the neighborhood.
2. Aggregate the feature information from sampled neighbors.
3. Based on the aggregation, we perform graph classification or node classification.



On each layer, we extend the neighborhood depth *K*, resulting in sampling node features K-hops away. This is similar to increasing the receptive field of classical convnets. One can easily understand how computationally efficient this is compared to using the entire neighborhood. That concludes the forward propagation of GraphSage. The key contribution, though, of the GraphSage paper is how they actually trained the model. The authors actually proposed two basic ideas:

1. Train the model in a fully unsupervised way. This can be done by using a loss function that enforces nearby nodes to have similar representations and disparate nodes to have distinct representations.
2. We can also train in a supervised manner using labels and a form of cross-entropy to learn the node representations

The tricky part is that we also train the aggregation function alongside our learnable weight matrices. The authors experimented with 3 different aggregation functions: a) a mean aggregator, b) an LSTM aggregator and c) an argmax-pooling aggregator. In all 3 cases, the functions contain trainable parameters that are learned during training. This way the network will teach itself the "correct" way to aggregate the features from the sampled nodes.

3. Model Training (on all_gene_disease_association)

In GNN, the input is a graph with nodes, and the output of it would be a graph with nodes with vector representation (how the node belongs in the context of the graph by some knowledge about node and neighbors iteratively in many steps to expand this knowledge). The neural network here has the responsibility to obtain this knowledge by updating weights in the network. So the only challenge in this part would be understanding what is appropriate input for neural networks, and the rest of the load of the work would be leveraged on the power of the neural network.

After creating the graph I need to transform it into a simple representation to be understandable for my code and nothing could be better than an adjacency matrix(not considering space cost by dense matrix) which will give information about the connection of nodes to the others and additionally they correspond to their label (y). at step=0 I consider the feature of each node is number of its neighbors.

3.1. GCN:

The first neural network to be evaluated is GCN which by design choice , has 2 hidden layers and one output layer. The convolutional layers for the graph are implemented by $GCNConv$ by the $torch\_geometric\ library$. The aim is to run this model on main dataset (All_gene_disease_assosiation) . However, I may consider the produced graph might be unbalanced so I train also on a slightly balanced version of the graph to compare the results. So in the following tables p would denote for the portion of nodes with label equal to 1 over the number of nodes of the graph.

My deduction : The choice of disease might be important. some models predict good on a disease but not good on another chosen disease

Plus in the following tables, other metrics are zero, which were expected to be, since the graph is still unbalanced. Dealing more with unbalanced graphs is done actually with the next dataset. So results here are not so interesting numerically in slightly balanced version

Disease	Asthma	Asthma (slightly balanced)	Diabetes	Diabetes (slightly Balanced)
p	0.11	0.15	0.12	0.15
Test accuracy	0.89	0.85	0.87	0.84
Test precision	0	0	0	0
Test F1-score	0	0	0	0
Recall	0	0	0	0
TP FP	0-0	0-0	0 – 0	0-0
TN FN	5129-656	4046-705	5042-743	4022 -743
Training time	2.68 s	1.86 s	1.92 s	1.8

If I balance the graphs more :

Disease	Asthma	Diabetes
p	0.37	0.41
Test accuracy	0.61	0.40
Test precision	0	0.41
Test F1-score	0	0.57
Recall	0	0.94
TP FP	0 – 0	698-1016
TN FN	1081-786	23-42

Comparing to the next model, it is interesting to see while GAT is slower , but if we have a balanced graph it will perform better in term of also other metrics, while in GCN is still zero.(at least for one disease)

3.2. GAT.

The second model is GAT. The input(graphs) are the same as the previous model. After training the model with saved graph models from the previous step (since making the graph is a time-consuming process) we will have the results (for the same diseases) as follows:

Disease	Asthma	Asthma (slightly balanced)	Diabetes	Diabetes (slightly Balanced)
p	0.11	0.15	0.12	0.15
Test accuracy	0.89	0.85	0.87	0.85
Test precision	0	0	0	0

Test F1-score	0	0	0	0
Recall	0	0	0	0
TP FP	0 – 0	0-0	0-0	0-0
TN FN	5129-656	4046-705	5042-743	4046-705
Training time	4.5	5.7	17	16

Note that this is the best performance of GAT achieved for dropout=0.7. changing dropout to other values will decrease the performance of the model in terms of metrics. Obviously from the table, values are quite close to metrics derived from previous model.
Plus I needed to train with more epochs, in other words, when I trained with 50 epochs, (similar to GCN), my accuracy was no more than around 20%

If I balance the graphs more :

Disease	Asthma	Diabetes
p	0.37	0.41
Test accuracy	0.61	0.41
Test precision	0.43	0.42
Test F1-score	0.06	0.58
Recall	0.03	0.99
TP FP	23-30	734-1038
TN FN	1051-655	1-6

3.3. GraphSAGE

Here are also training results for this model.(20 epochs only)

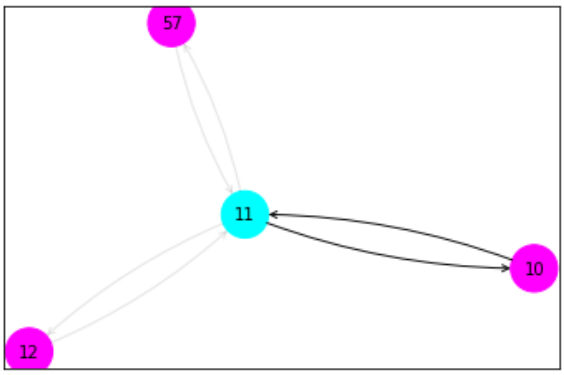
Disease	Asthma	Diabetes
p	0.11	0.12
Test accuracy	0.89	0.87
Test precision	0	0
Test F1-score	0	0
Recall	0	0
TP FP	0-0	0 – 0
TN FN	17101-2180	5042-743
Training time	18s	1.92 s

If I balance the graphs more :

Disease	Asthma	Diabetes
p	0.63	0.41
Test accuracy	0.66	0.6
Test precision	0.04	0.6
Test F1-score	0.06	0.09
Recall	0.02	0.05
TP FP	47-24	111 74
TN FN	3557-2133	3439-2305

3.4. Explainers

By using explainers, we want to know what are the crucial features or structures in the graph that affect the decisions of a neural network. The primary objective for GNNExplainer is to generate a minimal graph that explains the decision for a node or a graph. We can also apply explainers to the mentioned model results the following are the results of the Explainers. Which is done through the code. The result of this part can be useful for someone working on a specific disease to find a particular finding which is trained in notebook. The following is a result of Explainer on GCN for example , trained on reduced version of PPI dataset:



3.5. Comparision:

Despite this was a very small work to have a general conclusion on results, but based on my reading and results of the trainings it can be said that GCNs are much more computationally effective but GCN is heavily dependent on the structure of the graph (because of coefficient cij).However, this didn't impact the result of our models .

In these models, all results are similar if the hyper parameters are chosen wisely, for example, the value k in Graphsage and dropout in GAT as mentioned before. GraphSage on the other hand is good in evolving graphs later with unseen nodes. Moreover, GAT is too slow, and if the graph grows in size it may take more and more time to train the model.

If we perform data balancing approach, metrics change as mentioned and become more interesting to evaluate and compare models, but still , it is not easy to say what model is better than others. The results are available, approach is clear, and one may pick a model based on preference resulted from evaluations.

4. Model Training (on curated_gene_disease_association):

Changing this dataset will change our perception of how genes are connected, so we expect to have different results if we train on another this dataset. For this purpose we should create the graphs again, in this case, I firstly wanted to consider only the main version of the graphs and will compare the results with the trained models on the main version of graphs, but after realizing it is too much unbalanced I decided to work on balanced version. Because since p will be too small, predicting metrics wouldn't be a challenging task. All metrics would be near zero except accuracy which will be near 1. So I will work only on the balanced version, where some zero-labeled nodes are dropped randomly. However for diabetes number of one-labeled nodes where only very few

(less than 10 nodes). So in order to be able to check the real performance of unbalancing the data, I decided to work on another disease.
Doing so, GAT did perform better in term of accuracy both on balanced and unbalanced graph, however GCN performance decreased in accuracy from around 0.99 to as follows:

```
train time: 2.153313398361206 seconds
Test Accuracy: 0.17234226447709594
test precision: 0.003953391686081886
Test f1 score: 0.007874015718698502
Test recall: 0.949999988079071
 #### confusion matrix test:
TP 19 FP 4787
TN 978 FN 1
```

It is because it needed to be trained more epochs , by doing so, accuracy increased and other metrics decreased to zero as it was expected. The same is true for graphs when it is trained on an unbalanced graph. So I deduce if the unbalancing is done wisely it can help to obtain more understandable results. More important, data balancing can be important, but we should take into account here must be at least some useful information. – I cant remove all nodes! In previous dataset, we were given some 1 labeled nodes, so data balancing could help, but in dataset of this part , at least with these diseases, data balancing didn't have magical power.

About Explainers, Obviously, since we obtained models that are not better than previous models obtained from the previous dataset, we cannot expect to have better explainer models. however, they can still be applied on these models.

5. Conclusion:

The graph NN seems mathematically more difficult to understand concerning the node2vec model but it is faster since classification, and embedding is all done by NN and the message passing by the use of convolutional NN. GNNs are not dependent on parameters such as p and q that we have in node2vec. Consequently, maybe node2vec might not be appropriate for very large data with many features, GNN is interestingly faster and it can be extended to be used for explainers by the use of GNNExplainer.

Definitely, the performance of graph neural networks cannot be said to be good or bad with respect to others with comparing on one type of study (gene-disease for example) on only two disease. Interestingly, there are papers that targeted the evaluation of different GNNs on different tasks. However, based on this comparision, I can say at least on this small study there are some points to be considered

1. Unbalanced data can impact the result and have effect on metrics.
2. balancing data may need to be done in a wise approach so no useful information is lost
3. In this study, even if a model doesn't perform well, by training them more they can output more or less same results
4. Training more is spending more time, in this case was seconds, but if the graph is larger, might be considerable. In particular GATs are fairly computationally efficient.(wrt node2vec) but not faster than gcn (this fact can be confirmed from the test results)