Benchmarking Graph Convolutional Networks and Graphsage

Samidha Verma [*]
MS(Research) Student
IIT Delhi
csy207575@cse.iitd.ac.in

Sai Kandregula
MS(Research) Student
IIT Delhi
csy207570@cse.iitd.ac.in

ABSTRACT

Over the last few years, Graph Neural Networks are finding applications in diverse fields due to their increased performance which is made possible by novel architectures and significant advancements being applied on the ubiquitous graph structured data. In this paper, we will try to better understand and shed some light on the Graph Convolutional Network and GraphSAGE architectures through benchmarking experiments on three different datasets: Proteins, PPI and Brightkite for three different tasks: Pair-wise node classification, link prediction and Multi-class node classification. Finally, we try to explain our experiment results with suitable explanations. We have made our code[1] and results public in order to facilitate further exploration on other architectures as well as make our results reproducible for some other experiments.

1. INTRODUCTION

Graph data is ubiquitous and finds applications in social computing, natural science(study of protein structures, molecules, etc.), knowledge graphs and many others. Graph neural networks are gaining popularity as they can better leverage and extract information from graph like data as they propagate information guided by the graph structure. Therefore due to their good performance and high interpretability, we would perform this benchmarking exercise to better understand variants of GNNs, namely Graph Convolutional Network and GraphSAGE both of which can perform inductive learning.

In section 2 of the paper, we have given a brief background about GCN and GraphSAGE architectures, more information about them may be gathered through the cited papers. In Section 3 we have given the description of the datasets and the tasks we would be performing in the benchmarking exercise along with the experimental setup. In Section 4 we have reported the results of our experiments and our analysis of those results which may better help to understand these GNN architectures.

2. BACKGROUND

Graph Convolutional Network

Similar to the ConvNets in computer vision, the GCN[1] architecture learns the permutation equivariance encoding, such as mean aggregation, by using local undirected graph structure and features of nodes in a transductive manner. The GCN design uses a single shared weight matrix per layer and deals with different node degrees by suitable normalization and thus giving equal importance to self connection and edges of neighbouring nodes. The assumption is that similar labelled nodes are more likely to be connected.

GraphSAGE

GraphSAGE[2] builds upon GCN by generalising the idea of message passing to inductive learning such that it learns aggregator functions of node and its local neighbourhood to effectively embed the new nodes without retraining the whole model. The learning stage involves mapping neighbourhood with aggregation to obtain neighbourhood embedding which is then concatenated with the existing node's embedding.

3. EXPERIMENTS

We consider the benchmarking on three different tasks:

- Pair-wise node classification,
- Link prediction,
- Multi-class node classification

All of our code has been written in Python using the Pytorch framework. We used P-GNN[3] code for preprocessing of datasets and evaluation of pair-wise node classification and link-prediction tasks. For link-prediction task on brightkite dataset we used Deep Graph Library(DGL) with Pytorch backend for scalability reasons however, the data preprocessing and evaluation is done along the lines of P-GNN[3] as well. For multi-class node classification we have implemented our GCN and GraphSAGE models using Pytorch framework.

3.1 Datasets

3.1.1 Proteins

Proteins dataset is from P-GNN[3] which contains 1113 graphs where each graph is a protein. Each node in the graph represents Secondary Structure Elements such as helices, sheets and turns; and edges connecting these nodes represent SSE's spatial neighbourhood or amino acid sequence neighbourhood[4]. Each node has 29 features and belongs to one of 3

[*]66.6% contribution.
[1]`https://github.com/Samidha09/COL868`

classes. Normalized Amino Acids indices for hydrophobicity, van der Waals volume, polarity and polarizability are applied to the sequence of each SSE node to derive 3-bin distribution. The average number of nodes and edges per graph are 52 and 197.

3.1.2 PPI

Protein-protein interactions depicts complex network of protein pair reactions which play an important regulatory and execution role in cellular and biological processes of all living organisms. So, studying them is necessary for understanding life functions, drug development and in prevention and diagnosis of diseases. However, the full network structure of all these interactions is harder to obtain or validate. Thus, there is a need for computational algorithms to predict these relations or labels(functions) of a unlabeled node, with the help of known ppi structures, which can then be validated in lab.
The PPI dataset we use is from P-GNN[3] repository and consists of 24 graphs each corresponding to a different human tissue. Each node denotes a protein which has 50 features including positional and motif gene sets, immunological signatures; and one or more of 121 labels from gene ontology sets[5]. The average number of nodes and edges per graph are 2360 and 52907.

3.1.3 Brightkite

Brightkite[6] is an undirected graph data set from location based social networking service, where locations are shared by each user from years 2008 to 2010. The data set contains 58,228 nodes and 214,078 edges in 4 connected components. Nodes and edges in the largest connected component are 56739 and 212945 with diameter of 16. The average clustering coefficient is 0.1723.

3.2 Tasks

3.2.1 Pair-wise node classification

This task is to predict if a pair of nodes are of same class label. Proteins dataset is used for this task. The features have been normalized so as to have zero mean and unit variance to facilitate in training. Then further graphs containing single isolated node have been removed from the dataset. Graphs having less than 25% pairs of nodes with the same labels have also been removed from the dataset as they are not very information rich.
Data splitting is done in 80:10:10 ratio for train, validation and test sets respectively. Nodes having the same labels are a part of the *positive labelled set*. *Negative labelled set* is sampled using pair of nodes not in the positive labelled set for each graph in train, validation and test sets.
GCN and GraphSAGE models with 2 layers and a learning rate of 0.01 have been used for a fair comparison in performance. Evaluation has been done using metrics: micro averaged precision, recall and F1-score and ROC-AUC score averaged over 5 runs in the tables. Binary averaged precision, recall, f1 score and ROC-AUC are reported in plots of training time vs metrics.
Our models use binary cross-entropy with logits loss as the objective function. We ran both models for 500 epochs.

3.2.2 Link Prediction

In this task, we try to infer unobserved or missing links based on the links from which we trained our GNN. The GCN and GraphSAGE architectures relies on the assumption that connected nodes are likely to share same label.

3.2.2.1 PPI.

In this task, we used PPI dataset for link prediction using GCN and GraphSAGE models. We ran both models for 600 epochs with learning rate of 0.01 using Adam optimiser. The models use binary cross-entropy with logits as the loss function and ReLU as activation function between layers, whereas sigmoid as the output activation function. The data is split in the ratio of 80% for training, 10% for validation and 10% for testing and we constructed the equal number of negative edge samples from the graph. The results of metrics: binary averaged precision, recall, f1 score and ROC-AUC are reported as plots in figures 1 and 2. Micro averaged results are reported in the tables in order to compare with results of other teams.

3.2.2.2 Brightkite.

In this task, we used Brightkite dataset for link prediction using GCN and Graphsage models. We ran both models for 100 epochs with learning rate of 0.01. The models use binary cross-entropy with logits as the loss function and ReLU as the activation function.
The data is split in the ratio of 90% for training and 10% for testing and we constructed negative edge samples, which are 5 times the number of positive edge samples, from the graph using uniform sampling. This is done because the number of negative edges in the graph is more, and hence a bias towards the negative class is appropriate. Moreover, since this graph has a huge connected component which had 56739 nodes and 212945 edges, that couldn't be loaded on Colab with it's RAM limit of 12GB being exceeded. We tried to run the P-GNN code at first but due to all pair shortest path computation for P-GNN anchors during data loading, it was inefficient as well and although it worked for the smaller graphs, it failed for such a large component. In that case, we wrote the link-prediction task using DGL which is faster than Pytorch Geometric and further added a heuristic of **neighbourhood sampling** for computing embeddings of a node in a particular layer by using embeddings of only 6 neighbours instead of all possible neighbours from the previous layer while aggregating using inbuilt sampler[7] in DGL. This setup ultimately helped in being able to scale the code for this dataset.

GCN and GraphSAGE models with 2 layers have been used for a fair comparison in performance in both the link prediction tasks.

3.2.3 Multi-class node classification

In this task, we predict all the class labels of every node. We use PPI data for this task and do 5-fold cross validation such that the graphs were randomly divided each time into training, validation and testing sets in the ratio 17:5:2. We set the learning rate at 0.001 to conduct ablation studies on different model settings for training and obtained metrics averaged over 5 fold cross validation.The models are evaluated with micro averaged F1-score, precision, recall and ROC-AUC metrics on validation and test data sets. We have conducted the following 3 experiments for this task in

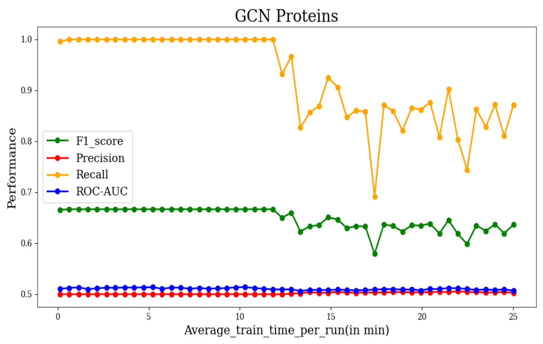

Figure 1: GCN performance vs training time

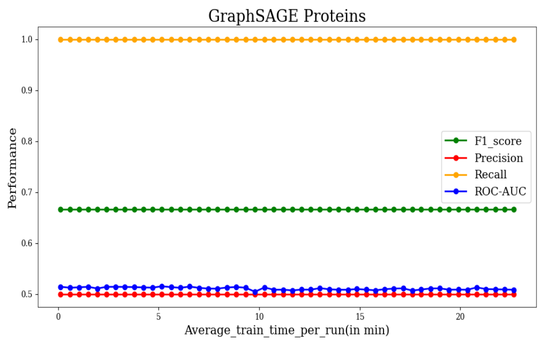

Figure 2: GraphSAGE performance vs training time

order to get an overview of how the GCN and GraphSAGE models perform when different parameters are tuned.

3.2.3.1 Experiment 1.

We trained GCN and GraphSAGE models with hidden_dim = 256, epochs = 100 while varying the number of layers {2, 3, 4, 5} and the activation function {ReLU, Sigmoid}. For GraphSAGE we experimented with different aggregators such as {Add, Max Pool, Mean Pool}. Standard GCN architecture from Kipf and Welling[1] was used for GCN model. The obtained metrics on test data set were averaged over the 5-folds and reported in Tables 4 and 5.

3.2.3.2 Experiment 2.

In this we studied the training time effect on performance with the best model variants observed on validation set from experiment 1. The settings are GraphSAGE model with 3 layers, ReLU activation, Max Pool aggregator and GCN model with 2 layers and ReLU activation function. We varied the epoch from 100 to 600 in steps of 100 and plotted the metrics with respect to average training time per fold.

3.2.3.3 Experiment 3.

In this setting, we conducted a study on how the embedding dimensions affect the model's performance. We vary embedding dimensions as {64, 128, 256, 512} and test the GCN and GraphSAGE model variants with best performance on validation set from experiment 1. We ran for 300 epochs considering this as a reasonably good setting from experiment 2.

4. RESULTS AND ANALYSIS

4.1 Pairwise Node Classification

The performance results of GCN and GraphSAGE on Proteins dataset with different average training time(in minutes) are reported in figures 1 and 2. Averaged results over 5 runs and model performance comparison is done in table 1

- We observed that GCN and GraphSAGE models couldn't distinguish between positive and negative examples well in this task and perform poorly. Here positive examples are the node pairs with same labels, and negative examples are node pairs with different labels.

Table 1: Pair-wise node classification on Proteins dataset. $L = 2$, $h_{dim} = 32$, $act = ReLU$, $lr = 0.01$

Model	ROC-AUC	Precision	Recall	F1-Score
GCN	0.495	0.506	0.506	0.506
GraphSAGE	0.499	0.500	0.500	0.500
Node2Vec	0.502	0.513	0.513	0.513
Deepwalk	0.504	0.509	0.509	0.509
GAT	**0.896**	**0.829**	**0.829**	**0.829**
P-GNN	0.811	0.803	0.576	0.647

The main reason could be the way the Proteins data is constructed, as seen from the fig 3, different types of SSE nodes within the spatial neighbourhood or amino acid sequence neighbourhood can be linked in the output proteins graphs. Now, with the GCN and GraphSAGE models being message passing architectures depending on aggregating neighbourhood information, in the Proteins networks we could not effectively use the position/location of the nodes as the models could embed different labelled nodes into similar points in embedding space.
For example, it could be the case that in this real world proteins network, a pair of nodes residing in various parts of graph could have similar local neighbourhood structure for which the GCN and GraphSAGE embeds them into same point in embedding space [3].

- We also observed low binary averaged precision and high binary averaged recall for both models with most of the predictions being 1 in this task. It indicates the models are placing too much emphasis on reducing false negatives but at the cost of making too many false positives.

- When we run it with 3 layers, the model performance deteriorated further. This drop may have been assosciated with vanishing gradients in back-propagation, overfitting due to the increasing number of parameters, and oversmoothing - where embeddings would become similar values after repeated convolutions.

- The models worked well for this task with a different dataset - **Communities**, which further supports our above points that the poor performance in this task is mainly attributed to the input graphs' structures.
Communities dataset is a connected caveman graph [8] with 1% edges randomly rewired, with 20 communities where each community has 20 nodes. Each node

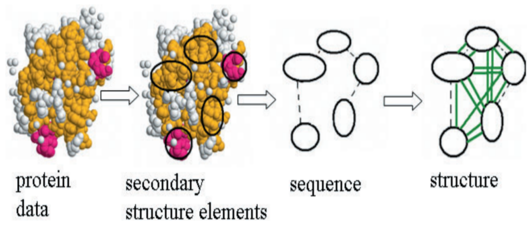

Figure 3: Proteins: Graph construction from proteins data bank [4]

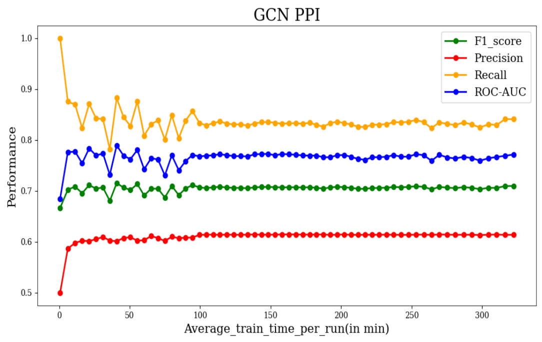

Figure 4: GCN performance vs training time on PPI

is labeled with the community it belongs to. Communities usually have a property of homophily. Since GCN and GraphSAGE model the homophily principle by propagating features and aggregating them within various graph neighborhoods via different mechanisms like mean pool, max pool, lstm and add, they performed well on this dataset. We observed F1-score, precision, recall and ROC-AUC as (0.839779, 0.72381, 1.0, 0.989436) and (0.875576, 0.778689, 1.0, 0.980679) for GCN and GraphSAGE respectively. Thus we observed even GCN and GraphSAGE can perform well in pairwise node classification tasks, on datasets that have homophily and non-similar local neighbourhood structures for different nodes.

4.2 Link Prediction

The link prediction performance results of GCN and GraphSAGE on PPI dataset are reported in figures 4 and 5. Averaged results and models comparison is done in table 2

4.2.1 PPI

Table 2: Link Prediction on PPI dataset.
L = 2, h_{dim} = 256, act = ReLU, lr = 0.01

Model	ROC-AUC	Precision	Recall	F1-Score
GCN	0.772	0.655	0.655	0.655
GraphSAGE	0.790	0.667	0.667	0.667
Node2Vec	0.944	0.957	0.929	0.943
Deepwalk	**0.972**	**0.956**	**0.989**	**0.972**
GAT	0.902	0.822	.822	0.822
P-GNN	0.784	0.730	0.701	0.715

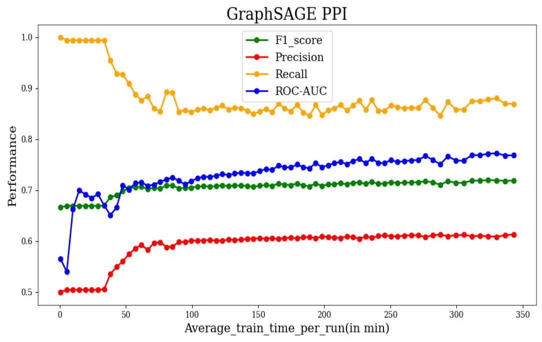

Figure 5: GraphSAGE performance vs training time on PPI

We ran link prediction on PPI dataset for 600 epochs for 5 runs and reported the average metrics on test set.

- Both GCN and GraphSAGE models' performance improved even till the end of 600 epochs, but the relative improvement remained low during the last epochs suggesting a plateau in performance.

- We observed that GraphSAGE performed slightly better than GCN. While GCN got AUC of 0.772, GraphSAGE attained 0.790.

- We also tested both GCN and GraphSAGE models by varying the number of layers to 3 and 4, but the performance is better with 2 layer model whereas it dropped by 5% and 15% with 3 and 4 layer models respectively. This indicates that the 2 hop neighbourhood gives enough information considering the fact that protein structured graphs may have a smaller diameter, and hence covering farther neighbourhoods might not give any useful information, rather cause oversmoothing of embeddings.

- The GAT model perfomed better in this compared to GCN and GraphSAGE as the aggregators are not able to give suitable consideration to the importance of each neighbour, the GAT during backpropagation perhaps gives better attention weights to those neighbours which occur frequently in multiple random walks from the source node. So, the GAT is a suitable model for this task in estimating the likelihood of a link between a pair of nodes with its attention learning mechanism, whereas GCN and GraphSAGE performs moderately.

- Even though GraphSAGE performed very well under multi-class node classification, it could not achieve similar performance in link prediction task on ppi dataset. This may be because it doesn't take into consideration the closeness of the nodes between whom the link has to be predicted through some weight mechanism and such an inference may prove more useful in this task. For node classification, mostly the node features may play a bigger role. For a similar reason we feel that Node2Vec and Deepwalk perform better than GCN, GraphSAGE and GAT because it better covers the closeness of a node to its neighbours via random walks.

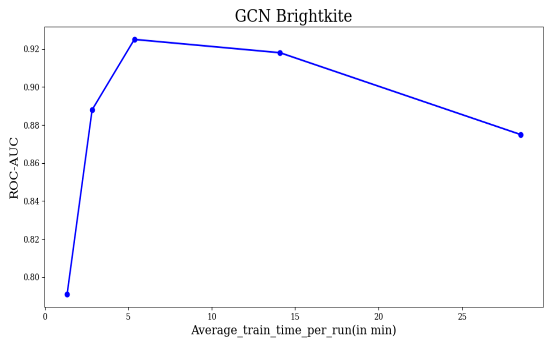

Figure 6: GCN performance vs training time on Brightkite

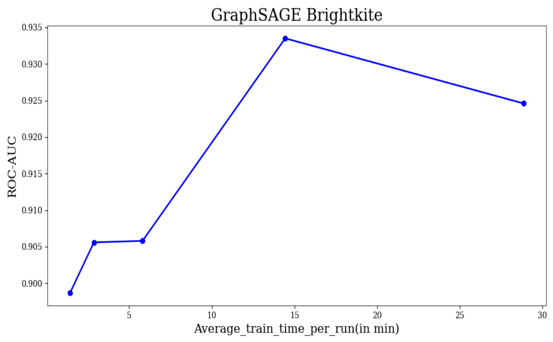

Figure 7: GraphSAGE performance vs training time on Brightkite

4.2.2 Brightkite

ROC-AUC scores averaged over 5 runs for link prediction on Brightkite dataset are reported in figures 6 and 7. Average ROC-AUC score and comparison with other models are done in table 3

We ran link prediction task on brightkite dataset for 5 runs, and have reported the average ROC-AUC score of test sets achieved in those runs. GCN and GraphSAGE variant with 2 layers have been used for this experiment.

- Both GCN and GraphSAGE models' performance kept improving till 50 epochs after which it started degrading perhaps due to overfitting on training data.
- Both the models have performed well on this task. GCN and GraphSAGE models produced an ROC-AUC score of 0.925 and 0.934 on the test set respectively.

Table 3: Link prediction on Brightkite dataset.
$L = 2$, $h_{dim} = 32$, $act = ReLU$, $lr = 0.01$, num_sampled_neighbours=6

	ROC-AUC
GCN	0.925
GraphSAGE	**0.934**
Node2Vec	0.696
Deepwalk	0.886
GAT	0.891
P-GNN	0.795

- It was also observed that GraphSAGE model performed better with relatively less amount of training, as compared to GCN. Hence, if the dataset is very large, for faster convergence of loss with less amount of training, GraphSAGE would be an appropriate choice. This may be because GraphSAGE concatenates the neighbour embeddings first and hence gathers more information quicker as compared to GCN which does by simply adding up the per neighbour normalized embeddings from the neighbourhood.
- Use of different aggregators, namely max pool and mean pool did not give any significant difference in performance of GraphSAGE model. This may be because in the preprocessing step where the latitude and longitude features were transformed to log scale so that logistics are not thrown off by big counts. Therefore, features of all neighbours in all dimensions would be of same scale and taking max pool wouldn't give any extra structural information over mean pool. Also a larger value of the latitude and longitude features actually had no importance over the others which had smaller value, they only gave positional information where friends must likely check in the same location at the same time. So whether choosing the larger valued feature or taking the mean, would not actually give the model any information apart from an actual location or the mean of locations of neighbours which would intuitively be same or close to the actual location considering the fact that friends may have checked in nearby or same place at the same time.
- Further, since we used neighbourhood sampling due to scalability issues arising because of the largest connected component with 56739 nodes and 212945 edges present in Brightkite, we tested with $4, 6$ as number of neighbours sampled from the previous layer, and sampling with 6 nodes performed better due to exploration of comparatively larger neighbourhood resulting in more information per node through message passing.
- We also tested the models by varying the embedding dimensions as 16, 32 and 32 performed better due to retention of more information.
- The results we have reported are for a split of 90:10 between train and test data. In order to ensure that the results are not too optimistic due to the small size of the test data, we also tested with 80:20 split between train and test data, and achieved similar results as reported.

4.3 Multi-class Node Classification

The ablation study results of GCN and GraphSAGE in multi-class node classification with various experiment settings follows: Experiment 1 results are reported in tables 4 and 5.

We tested the models with multiple learning rates 0.1, 0.05, 0.01, 0.001 and observed that at 0.1 learning rate the loss kept fluctuating and did not converge, for 0.05 and 0.01 the loss reduced for some epochs after which it wouldn't reduce and started fluctuating. 0.001 was the best learning rate observed and has been used for the below mentioned experiments.

Table 4: GCN metrics: Multi-class node classification on PPI

#Layers	**Activation**	**ROC-AUC**	**Precision**	**Recall**	**F1-Score**
2	Relu	**0.679**	**0.789**	**0.569**	**0.661**
3	Relu	0.647	0.727	0.549	0.624
4	Relu	0.625	0.722	0.506	0.595
5	Relu	0.607	0.694	0.496	0.578
2	Sigmoid	0.544	0.675	0.385	0.489
3	Sigmoid	0.534	0.672	0.364	0.472
4	Sigmoid	0.529	0.675	0.354	0.464
5	Sigmoid	0.526	0.677	0.344	0.456

Table 5: GraphSAGE metrics: Multi-class node classification on PPI

Aggregator	**#Layers**	**Activation**	**ROC-AUC**	**Precision**	**Recall**	**F1-Score**
Add	2	Relu	0.774	0.827	0.713	0.766
Add	3	Relu	0.501	0.666	0.289	0.403
Add	4	Relu	0.500	0.664	0.289	0.403
Add	5	Relu	0.500	0.597	0.322	0.400
Add	2	Sigmoid	0.683	0.767	0.591	0.661
Add	3	Sigmoid	0.661	0.719	0.583	0.641
Add	4	Sigmoid	0.607	0.697	0.499	0.581
Add	5	Sigmoid	0.538	0.675	0.374	0.481
Max pool	2	Relu	0.870	0.880	0.844	0.862
Max pool	3	Relu	**0.889**	**0.899**	**0.864**	**0.881**
Max pool	4	Relu	0.880	0.897	0.851	0.873
Max pool	5	Relu	0.859	0.883	0.825	0.853
Max pool	2	Sigmoid	0.693	0.767	0.616	0.682
Max pool	3	Sigmoid	0.686	0.755	0.616	0.678
Max pool	4	Sigmoid	0.675	0.765	0.584	0.662
Max pool	5	Sigmoid	0.667	0.751	0.576	0.652
Mean pool	2	Relu	0.807	0.849	0.762	0.803
Mean pool	3	Relu	0.825	0.866	0.780	0.820
Mean pool	4	Relu	0.827	0.854	0.792	0.822
Mean pool	5	Relu	0.817	0.861	0.771	0.814
Mean pool	2	Sigmoid	0.617	0.726	0.504	0.594
Mean pool	3	Sigmoid	0.611	0.707	0.505	0.588
Mean pool	4	Sigmoid	0.568	0.678	0.431	0.523
Mean pool	5	Sigmoid	0.500	0.666	0.286	0.400

Table 6: GraphSAGE metrics: Multi-class node classification on PPI

Model	**ROC-AUC**	**Precision**	**Recall**	**F1-Score**
GCN	0.789	0.818	0.744	0.779
GraphSAGE	0.960	**0.957**	**0.953**	**0.955**
Node2Vec	0.673	0.635	0.463	0.536
Deepwalk	0.663	0.643	0.430	0.515
GAT	**0.970**	0.924	0.925	0.925
P-GNN	0.656	0.621	0.481	0.543

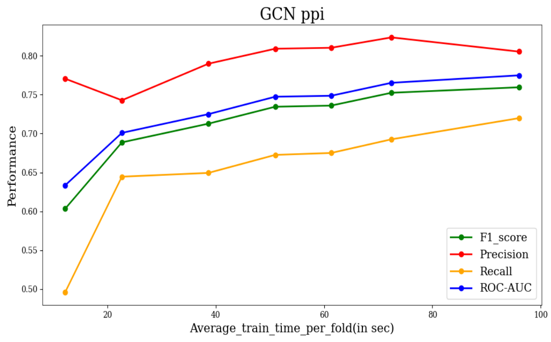

Figure 8: GCN performance vs training time on Multi-Class Node classification

4.3.1 Experiment 1:

Overall we observed that the best GCN model performs 20% worse on average compared to best GraphSAGE model, with GraphSAGE being superior on all metrics.

- **GCN**: The 2 layer GCN variant with ReLU activation function performed the best on all metrics in GCN model.

- **GraphSAGE**: The GraphSAGE variant with 3 layers and Max Pool aggregator gave best results in this experiment with both relu and sigmoid activation functions. This may be because the features of the nodes were mostly discrete and taking mean or sum would lead to loss in information. Taking Max Pool might retain the important feature structures. It can also be observed that if Add aggregator is used with relu activation, with increase in number of layers, there is significant drop in performance, this may further solidify the above reasoning as simply adding information from farther neighbours may lead to worst embeddings.

- Another observation is regarding the choice of the activation function. Relu activation led to better results as compared to sigmoid in the case of GCN as well as GraphSAGE. This may be because of vanishing gradients problem that is a disadvantage of sigmoid function. Even though the number of layers in the network are less, still aggregating information from the neighbours might have led to larger values, and hence the vanishing gradient problem may have arised. We feel that using batch normalization layers might lead to improvement in the results as they normalize the input so that it doesn't reach the outer edges of the sigmoid function

4.3.2 Experiment 2:

Experiment 2 results are reported as plots in figures 8 and 9.

- **GCN**:For GCN the performance was best at 600 epochs for both validation and test sets, after which it started to worsen.

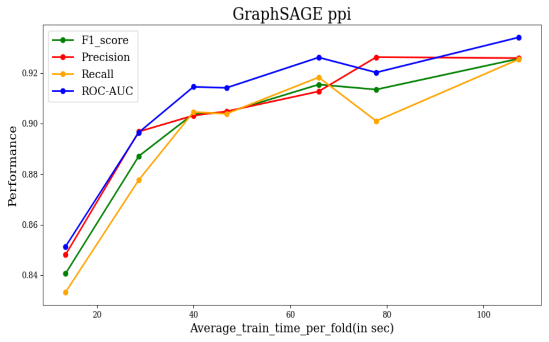

Figure 9: GraphSAGE performance vs training time on Multi-Class Node classification

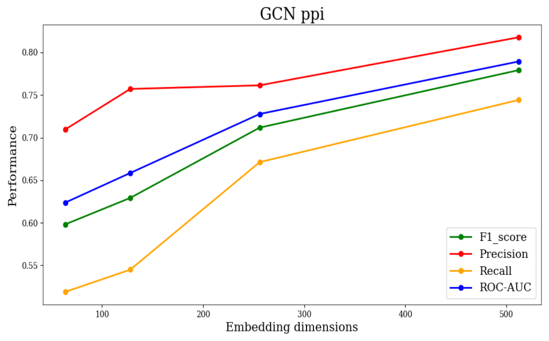

Figure 10: GCN performance with different embedding dimensions on Multi-Class Node classification

- **GraphSAGE**:We observed that the GraphSAGE model performance improved consistently until 500 epochs, but began to worsen after.

The main observation here was that the GraphSAGE model(best from experiment 1) gave much better performance at relatively lower number of epochs in comparison to GCN model(best from experiment 1). This may be because GraphSAGE concatenates the neighbour embeddings first and hence gathers more information quicker as compared to GCN which does by simply adding up the per neighbour normalized embeddings from the neighbourhood(also observed in Section 5.2.2). Even GraphSAGE with mean pool aggregator which is relatively closer to GCN aggregator may have performed better at far less number of epochs due to the same reason as well.
Experiment 3 results are plotted in figures 10 and 11

4.3.3 Experiment 3:

From figures 10 and 11, it can be observed that the performance of both GCN and GraphSAGE models increases with increased significantly with an increase in the number of embeddeding dimensions as expected because larger amount of information can be retained.This comes at the cost of increase in the training time that may lead to scalability issues in case larger datasets are used.

5. CONCLUSIONS

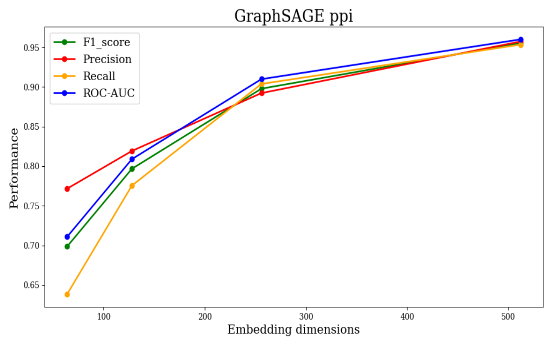

Figure 11: GraphSAGE performance with different embedding dimensions on Multi-Class Node classification

We conducted different experiments for benchmarking GCN and GraphSAGE models on tasks: Pairwise node classification, link prediction and Multi label node classification using metrics ROC-AUC, Precision, Recall and F1 scores. The ablation studies are also conducted in different tasks to find the suitable hyperparameters and to better understand the roles they play in the prediction. The observations made are supplied with suitable reasoning in each scenario. However, in general it is observed that both GCN and GraphSAGE perform better when number of layers are restricted to 2 or 3, after that their performance worsens maybe because of oversmoothing of embeddings after multiple convolutions. Increasing the hidden dimensions or the embedding dimensions further causes a boost in performance due to retention of more information. Further through neighbourhood sampling exercise we also conclude that wider networks perform better as the nodes can gather more information from more number of neighbours. Also, GCN and GraphSAGE architectures perform well when the graphs have homophily property. Finally, our github repository can be referenced for further experiments.

6. ACKNOWLEDGEMENTS

We would like to thank Professor Sayan Ranu, IIT Delhi for giving us the opportunity to learn and dig deeper into GNN models with these experiments. We would also like to thank Google Colab for the computational resources involved in the experiments.

7. REFERENCES

[1] Thomas Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907, 2017.

[2] William L. Hamilton, Zhitao Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[3] `https://github.com/JiaxuanYou/P-GNN`.

[4] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1):47–56, January 2005.

[5] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *CoRR*, abs/1707.04638, 2017.

[6] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: User movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, page 1082–1090, New York, NY, USA, 2011. Association for Computing Machinery.

[7] `https://docs.dgl.ai/en/latest/new-tutorial/L0_neighbor_sampling_overview.html`.

[8] Duncan J. Watts. Networks, dynamics, and the small-world phenomenon. *American Journal of Sociology*, 105(2):493–527, 1999.