APPLICATION OF GRAPH CONVOLUTIONS FOR SEMI-SUPERVISED NODE CLASSIFICATION

A Project Report submitted to the

Indian Statistical Institute, Kolkata

in partial fulfilment of the requirements of

The Seventh Summer School

on

Computer Vision, Graphics and Image Processing

by

Anannyo Dey

(Jadavpur University, Kolkata)

and

Soumyajit Roy Chowdhury

(KIIT, Bhubaneswar)

Utpal Sarkar

(Jadavpur University, Kolkata)

under the supervision of

Kushal Bose

SRF, ECSU, Indian Statistical Institute, Kolkata

Electronics and Communication Sciences Unit

Indian Statistical Institute

203, B T Road, Kolkata - 700108

PROJECT CERTIFICATE

This is to certify that the Project entitled "*Application of Graph Convolutions for Semi-Supervised Node Classification*" submitted by **Anannyo Dey**, Jadavpur University, Kolkata, **Soumyojit Roy Chowdhury**, KIIT, Bhubaneswar, and Utpal Sarkar, Jadavpur University, Kolkata to the Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata towards the partial fulfilment of the participation in the Sixth Summer School on Computer Vision, Graphics and Image Processing during May 31 – July 12, 2022. The entire project work is carried out by them under my supervision. The contents of this Project have not been submitted to any other Institute or University for the award of any degree or diploma.

Place: Kolkata

Date: July 11, 2022

Name of the Supervisor: Kushal Bose

Designation: Senior Research Fellow

Electronics and Communication Sciences Unit

Indian Statistical Institute, Kolkata

Kolkata - 700108

ABSTRACT:

Many important real-world datasets come in the form of graphs or networks: social networks, knowledge graphs, drug-interaction networks, etc. In recent years, Graph Neural Networks (GNNs) have become an indispensable tool to learn from graph data, and have been shown to perform well on various prediction tasks. We aim to solve Semi-Supervised Node Classification on 3 benchmark citation network datasets- Cora, Citeseer, Pubmed. GCNs work on Message Passing NNs that combine node feature aggregation and feature transformation using learnable weight matrix. We propose minor enhancements of hyper-parameter optimization. We present extensive results of our implementation of the 2-layer GCN in PyTorch, showing that our approach achieves improved test accuracy for Cora w.r.t. T. Kipf’s GCN paper and competitive results for other 2 datasets. We also experimented by introducing Deep Learning into the model by stacking up-to 1024 layers (progressing in powers of 2), in quest for a model that is more aware of the graph structure, but faced Over-smoothing issues and thus, analysed the drastic drop in accuracy.

1. Introduction

Extensive work has been done on CNNs and RNNs to deal with images and textual data. But in reality there's more abundance of structured datasets in form of graphs. Yet, until recently, very little attention has been devoted to the generalization of NN models to such structured datasets.

In recent years, Graph Neural Networks (GNNs) have opened a unique path to learning on data by leveraging the intrinsic relations between entities that can be structured as a graph. By imposing these structural constraints, additional information can be learned and used for many types of prediction tasks. With rapid development of the field and easy accessibility of computation and data, GNNs have been used to solve a variety of problems like node classification, link prediction, graph classification, prediction of molecular properties, node ranking and natural language processing.

In this work, we focus on the Semi-Supervised Node classification task using GNNs. We were motivated by the success of early GNN models such as GCN [1], and an increasing trend in improving the expressiveness of the model. These models are more suitable for homophily datasets, where nodes that are linked to each other are more likely to belong to the same class. GNN models combine feature aggregation and transformation using a learnable weight matrix in the same layer, often referred to as graph convolutional layer. These layers are stacked together with the non-linear transformation (e.g., ReLU) and regularization (e.g., Dropout) as a learning framework on the graph data, that has the effect of introducing powers of adjacency matrix (or laplacian), which helps to generate a new set of features for a node by aggregating neighbor's features at multiple hops, thus encoding the neighborhood information. The number of these unique features depends on the propagation steps or the depth of the model.

We also tried to gauge the performance of Deep GCN models, as we stacked up more layers, thereby increasing the number of hops for aggregation, in search of an improvement in test accuracy. We increased the number of layers (hop setting) in powers of 2, starting from 2, 4, 8, 16, 32, 64, 128, 256, 512, up until we reached 1024 layers.

2. Preliminaries

Let $G = (V, E)$ be an undirected graph with n nodes and m edges. For numerical calculations, graph is represented as adjacency matrix denoted by $A \in \{0, 1\}^{n\times n}$ with each element $A_{ij} = 1$ if there exists an edge between node v_i and v_j ,otherwise $A_{ij} = 0$. When self-loops are added to the graph then, resultant adjacency matrix is denoted as $\tilde{A} = A + I$. Diagonal degree matrix of A and $\tilde{A}$ are denoted as D and $D^{\sim}$ respectively. Each node is associated with a d-dimensional feature vector and the feature matrix for all nodes is represented as $X \in \mathbb{R}^{n\times d}$.

Conventionally, a simple GNN layer is defined as-

$$f\left(H^{(l+1)}, A\right) = \sigma(AH^{(l)}W^{(l)})$$

where $\hat{A} = \tilde{D}^{\frac{-1}{2}}\tilde{A}\tilde{D}^{\frac{-1}{2}}$ is a symmetric normalized adjacency matrix with added self-loops. $H^{(i)}$ represents features from the previous layer, $W^{(i)}$ denotes the learnable weight matrix, and σ is a non-linear activation function, which is usually ReLU in most implementations of GNNs. The cumulative aggregation of node's self-features with that of neighbors reinforces the signal corresponding to the label and helps to improve accuracy of the predictions.

Following the conventional GNN formulation using $\tilde{A}$, a simple 2-layered GNN can be represented as,

$$Z = f(X, A) = \mathrm{softmax}(\hat{A}\,\mathrm{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)})$$

2.1 Node Classification

Node classification is an extensively studied graph based semi-supervised learning problem. It encompasses training the GNN to predict labels of nodes based on the features and neighborhood structure of the nodes. Using Eq. (2), GNN aggregates the features of two hops of neighbors and outputs *Z*. Softmax function is applied row-wise, and cross-entropy error is calculated over all labeled training examples. The gradients of loss are back-propagated through the GNN layers. Once trained, the model can be used to predict labels of nodes in the test set.

3. Experiments

3.1 Datasets

We consider three citation network datasets: Citeseer, Cora and Pubmed (Sen et al., 2008). The datasets contain sparse bag-of-words feature vectors for each document and a list of citation links between documents. We treat the citation links as (undirected) edges and construct a binary, symmetric adjacency matrix A. Each document has a class label. For training, we only use 20 labels per class, but all feature vectors.

Datasets	Nodes	Edges	Features	Classes	Label Rate
Cora	2,708	5,429	1,433	7	0.036
Citeseer	3,327	4,732	3,703	6	0.052
Pubmed	19,717	44,338	500	3	0.003

Table 1: Dataset statistics, as used by T. Kipf et al.

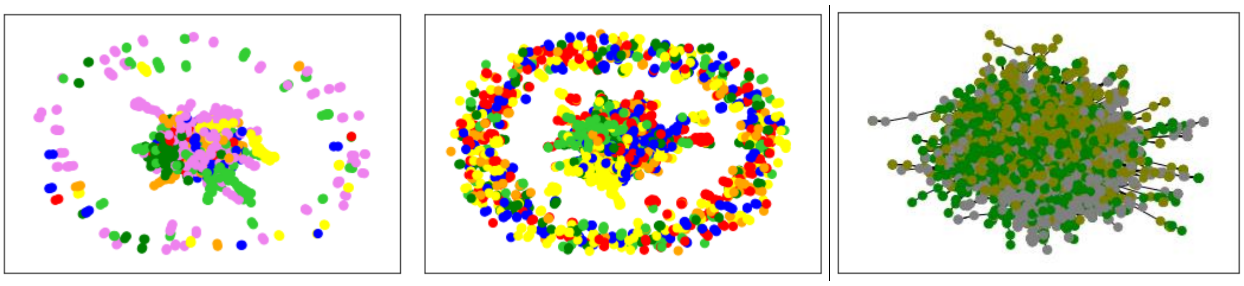


Figure 1: Visualizing Datasets

3.2 Hyperparameter settings

We train a two-layer GCN in PyTorch (instead of TensorFlow as in the paper) and evaluate prediction accuracy on a test set of 1,000 labeled examples. We closely follow the experimental setup in T. Kipf et al. (ICLR 2017) and choose the same dataset splits as in Yang et al. (2016) with a validation set of 500 labeled examples for hyperparameter optimization, but their labels are not used for training. We train all models for a maximum of 200 epochs (training iterations) using Adam optimizer (Kingma & Ba, 2015) and a Cross-Entropy loss function, with a learning rate of 0.01. We used the following sets of hyperparameters: 0.5 (dropout rate), $5 \cdot 10^{-4}$ (L2 regularization) and 16 (number of hidden units).

4. Our contributions

Our contributions are 2 fold-

i. PyTorch still uses old strategies for default weight initialization when creating Conv/Linear Layers to maintain backward compatibility (As of July 2022, this issue is still open on Github). But we know good initialization matters, as wrong initialization of weights can lead to vanishing or exploding gradients. For using ReLU, we initialized weights with Kaiming He initialization and set the biases to zero (2014 ImageNet winning paper from Microsoft). Using the 'fan_in' mode ensures that the data is preserved from exploding or imploding.

ii. We also tried to gauge the performance of Deep GCN models, as we stacked up more layers, thereby increasing the number of hops for aggregation, in quest of an improvement in test accuracy. We increased the number of layers (hop setting) in powers of 2, starting from 2, 4, 8, 16, 32, 64, 128, 256, 512, up until we reached 1024 layers.

5. Results

5.1 Semi-supervised node classification

We have summarized experiments by showing the best results of all our enhancements for all the datasets in Table 2. Reported numbers denote classification accuracy on test dataset in percent. Results for the other baseline method are taken from Kipf & Welling (ICLR 2017), and we were able to succesfully achieve *improved* accuracy for Cora dataset, and competitive results for the other 2.

Model	Cora	Citeseer	Pubmed
GCN Paper	81.5	70.3	79
Ours	**81.7**	69.7	78.6

Table 2: Comparison of test accuracy on 3 citation networks, with that of reported in T. Kipf et al. (2017)

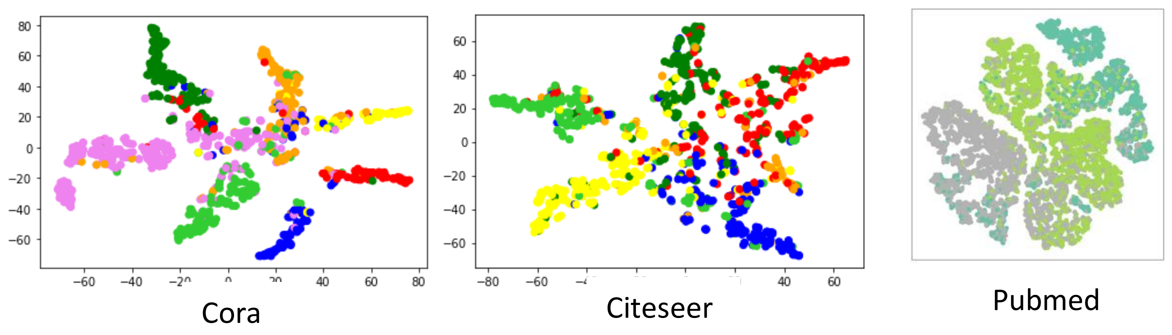
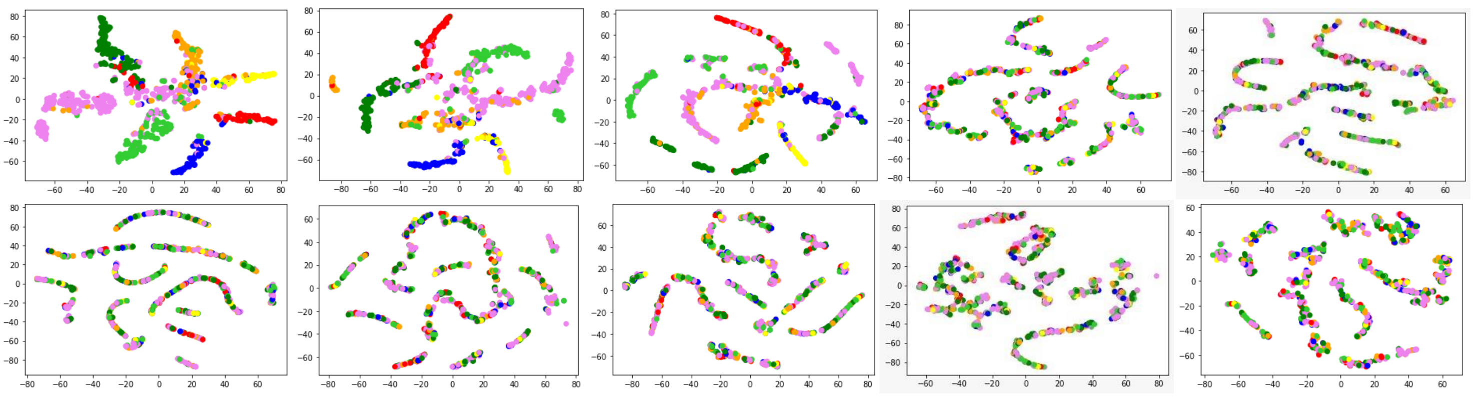


Figure 2: t-SNE plots of Node embeddings after applying a 2-layer GCN, shows compact clustering.

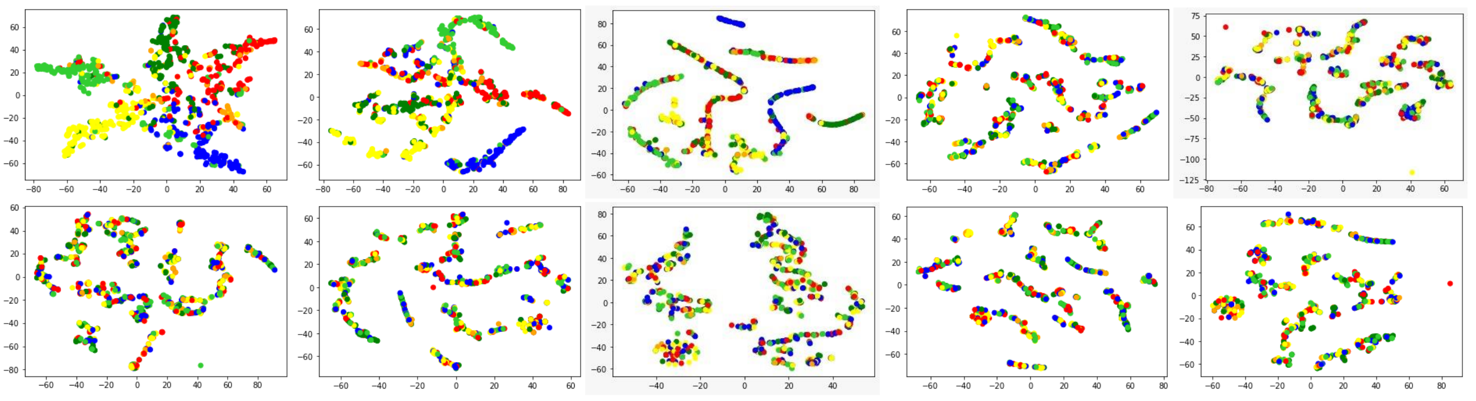
5.2 Performance evaluation of Deep GCNs

We ran additional experiments on our model for Cora and Citeseer datasets, with hop values set to 4, 8, 16, 32, 64, 128, 256, 512, 1024 with all features as input. We plot the 10 node embeddings for each of the datasets, and observed that as we stacked up more layers, the graph gradually collapsed on itself. MPNN thus faces limitations as depth increases, leading to indistinguishable representations of nodes in different classes.

This phenomenon of ending up with similar embeddings for nodes that don't have the same label, resulting in mislabeling them, is known as *Oversmoothing*.

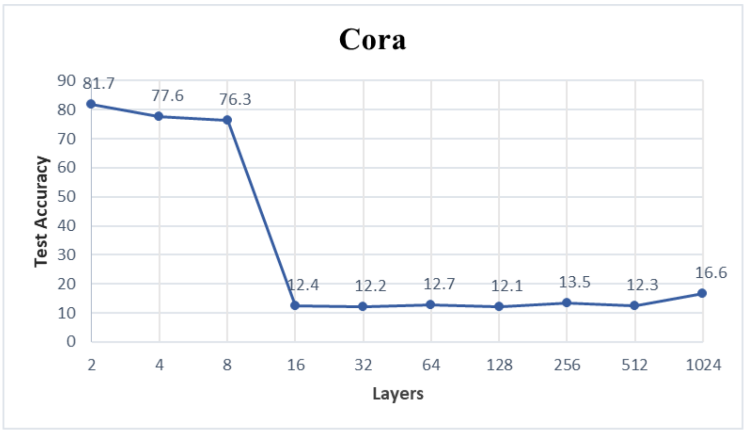


(a) Cora dataset



(b) CiteSeer dataset

Figure 3: Graph Node embeddings for hop setting of 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 for (a)Cora and (b)Citeseer datasets.



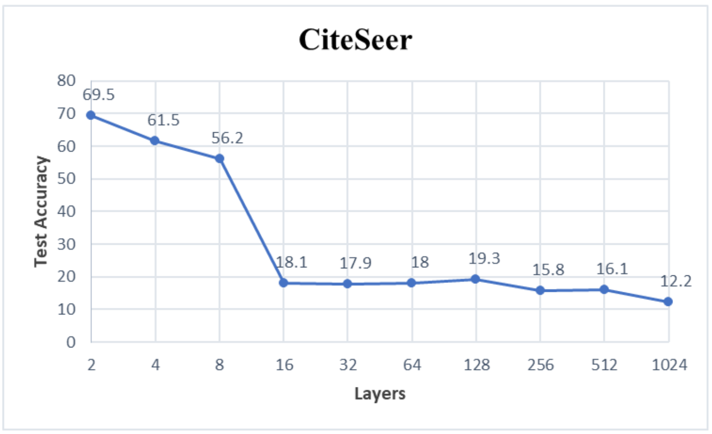


Figure 4: Performance of the model with hop setting in powers of 2 (presented in log scale)

Many GNN models suffer from this Oversmoothing problem when the number of hops for feature aggregation is increased. In this section, we evaluated the change in model's performance with increase in the hops for aggregation. We observed the drastic drop in test accuracy as we stacked up more layers, performing multi-hop neighbourhood aggregation.

6. Conclusions

In this work, we implemented a 2-layer GCN model using PyTorch in Google Colab to solve a Semi-Supervised Node Classification task on 3 benchmark citation networks- CORA, CiteSeer, PubMed, with minor enhancements for hyperparamaeter optimization that resulted in improved accuracy for Cora dataset and competitive results for the other 2. We also explored the performance of Deep GCNs. We ran extensive experiments with hop values set to 4, 8, 16, 32, 64, 128, 256, 512, and 1024, to investigate the drop in accuracy for both Cora and Citeseer datasets.

7. Future work

Solve the Over-Smoothing problem by-

i. Avoiding the creation of almost similar embedding between nodes.

ii. The model can be trained to learn to place low weights on higher hops.

8. References:

[1] T. N. Kipf, M. Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR (2017).

[2] Graph Convolutional Networks, T. Kipf, University of Amsterdam, http://tkipf.github.io/graph-convolutional-networks/

[3] Maurya, S. K., Liu, X., & Murata, T. (2022). Simplifying approach to node classification in graph neural networks. Journal of Computational Science, 62, 101695.

[4] Adaloglou, Nikolas. "How Graph Neural Networks (GNN) Work: Introduction to Graph Convolutions from Scratch | AI Summer." AI Summer, theaisummer.com, 8 Apr. 2021.

[5] Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In International Conference on Machine Learning (ICML), 2016.

[6] He, Kaiming, et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." *arXiv.Org*, arxiv.org, 6 Feb. 2015.