Attributed Network Representation Learning

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Anket Kotkar
(180101037)

under the guidance of

Dr. Sanasam Ranbir Singh

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

Acknowledgements

I would like to express my gratitude towards **Dr. Sanasam Ranbir Singh** for his timely guidance and encouragement to pursue the problem statement for my project with a novel approach. I would like to thank Ms. Anasua Mitra for her timely support towards my project. At last, I would like to thank Computer Science and Engineering department for providing the resources for computation.

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Attributed Network Representation Learning**" is a bonafide work of **Anket Kotkar** (**Roll No. 180101037**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Sanasam Ranbir Singh**

Associate Professor,

May, 2022 — Department of Computer Science & Engineering,

Guwahati. — Indian Institute of Technology Guwahati, Assam.

Contents

List of Figures **iv**

List of Tables **v**

Abstract **1**

1 Introduction **2**

1.1 Necessity of Analysis 2

1.2 New Revolutionary Way 3

1.3 Organization of Report 3

2 Literature Survey **5**

2.1 Random Walk based methods 5

2.2 Neural Network based models 6

2.3 Autoencoder based methods 7

3 Problem Definition **9**

3.1 Definitions 9

3.2 Motivation 10

3.3 Problem Statement 10

4 Proposed solution **11**

4.1 GraphBert Working 11

4.2 Modified Sampling 13

4.3 Conclusion 13

5 Experiments 14

5.1 Datasets and Learning Condition 14

5.2 Results 15

5.2.1 Cora 15

5.2.2 Citeseer 18

5.3 Reasoning for results on Citeseer 18

6 Discussion and Conclusion 22

References 23

List of Figures

2.1 Random Walk Methods 6

2.2 Neural Network Methods 7

2.3 DANE Architecture 8

5.1 Comparison between k and mean accuracy for different β on Cora 16

5.2 Original GraphBert on Cora 17

5.3 Comparison between k and mean accuracy for different β on Citeseer 21

List of Tables

5.1 Datasets . . . 14
5.2 Default settings . . . 15
5.3 Different Model on Datasets . . . 15
5.4 Statistics for Original GraphBert on Cora . . . 17
5.5 Statistics for modified GraphBert on Cora for beta = 0.5 . . . 18
5.6 Statistics for Original GraphBert on Citeseer . . . 19
5.7 Statistics for modified GraphBert on citeseer for beta = 0.5 . . . 20

Abstract

The considerable dependence of highly successful Graph Neural Networks on edges present in the network causes them to suffer from many well-known problems. To overcome this, GraphBert (Graph-based Bert) is proposed as a new model which distances itself from the links present in the network and uses the attention mechanism to learn the node representations. GraphBert uses subgraphs sampled in the initial stage of linkless batching to learn the representation of the node corresponding to the subgraph. Using linkless batching, GraphBert tries to overcome the significant dependence of the model on graph edges. However, GraphBert only considers the network structure while sampling the subgraphs. To overcome this, we propose to modify the sampling algorithm to include the node attribute proximity and test it on the node classification task on graph benchmark datasets. At last, we compare the results of the modified sampling GraphBert with the original GraphBert.

Chapter 1

Introduction

Many naturally occurring real-life scenarios, such as social media interactions, communication networks, citation networks or bio-molecules networks can be modeled as graphs. In such different contexts, a network between different entities can be visualised to capture relations or interactions along with the node attributes and get a more meaningful and informative picture of the system. The network provides a more detailed representation of relationships observed in a real life scenarios.

1.1 Necessity of Analysis

Relations between multiple types of entities play a vital role in many settings like social structures, information spread and social choices. Modelling these interactions along with the attributes information available as a network helps researchers to analyze the systems. Importance of network analysis is visible across different domains like specific feed in social networks, viable cures in medicine or recommendations in different service providers.

How the network is represented highly impacts the network analysis tasks. Using adjacency matrix to represent the network under study cause issues most of the time due the scale of real world networks. Quadratic complexity in the number of vertices often makes the matrix operations infeasible on the real world graphs. Furthermore, to learn the high-level constructs and highly non-linear network structure and attributes, matrix representation is not sufficient. To overcome these problems, recently, a new way called Network Representation Learning or simple words NRL, has gained importance.

1.2 New Revolutionary Way

NRL tries to represent nodes in the network using low dimensional vectors to preserve topological space proximity. Attributed NRL tries to preserve attribute space proximity along with topological space proximity. These low dimensional vectors are called embeddings. To preserve the topological or network structure proximity implies that nodes with similar neighbourhoods have similar vector representation. Euclidean distance or cosine similarity can be used to measure the similarity. These embeddings are fed as an input to different downstream tasks to make the computation practically possible.

The NRL field has seen outstanding advancements in learning algorithms. Initial methods in the NRL field were using matrix factorization which were replaced by better random walk methods. But all these only considered the network topology. Attributes and metadata associated with the network play a crucial role in how these networks evolve or are created apart from just the topological structure. With the growth of graph neural networks, node attributes were considered in the learning process along with the network structure. Using attributes have shown to improve the performance across the tasks and datasets. But this GNN suffer from learning problems like suspended animation and over smoothing. Recently more powerful transformer based proposed models use attention mechanism to learn superior representations. These models also overcome the problems faced by GNN models. We discuss this in detail in chapter 2.

1.3 Organization of Report

In this chapter, we provided an introduction to Network Representation Learning and the necessity of such a topic. For the rest of the report, an overview of the topics covered is as follows:

- Literature Survey

- Problem formulation
- Proposed solution
- Experimental settings and results

At last, we discuss the possible future directions for this work.

Chapter 2

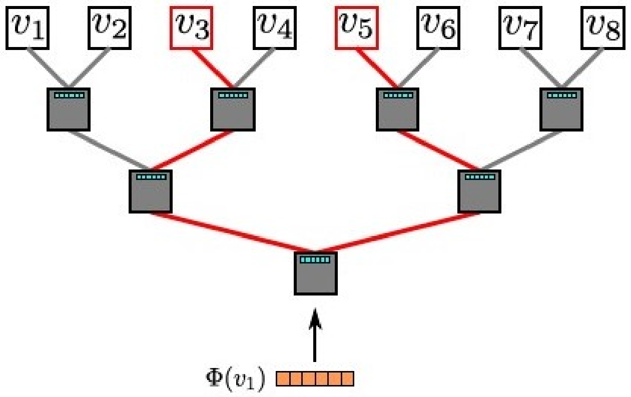
Literature Survey

Network Representation Learning field has taken giant leap recent times. In this chapter, we will go through some of the techniques.

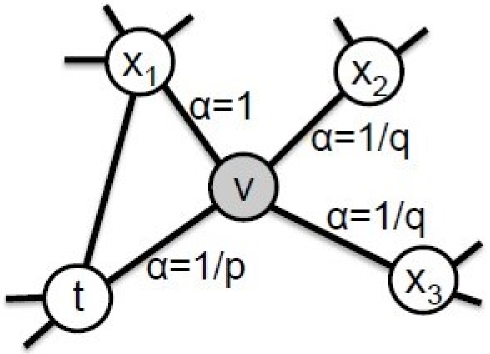
2.1 Random Walk based methods

Skipgram algorithm have given great results in NLP domain using word sequences to learn embeddings for the words in the corpus. Inspired from Skipgram, random walk-based methods like Deepwalk and Node2Vec bring that analogy in network domain by using graph as a vast corpus of nodes and using inter-relations to sample node sequences similar to word sequences.

Deepwalk[1] samples the random node sequences by considering the next node uniformly randomly from the node's neighbourhood currently under consideration. On these sampled sequences, Skipgram is applied to learn the node embeddings. For this, Deepwalk introduces a new hierarchical softmax function (Fig 2.1(a)) as the original softmax function in Skipgram is computationally very expensive. Nevertheless, Deepwalk is constrained by its rigid random walk sampling method, which is unsuitable for all the networks. Node2Vec[2] introduces an innovative idea of flexible sampling node sequences in the walks using tunable parameters. In Node2Vec, 2^{nd} order random node sequences are created using the modified transition probability between nodes. In this biased random walk setup, the modified transition probability from node u to node v also depends on the distance between the previous node in the random walk and node v (Fig 2.1(b)). The edge weights are the coefficients of the original transition probability. Node2Vec too uses the Skipgram model to node sequences and generate node embeddings using negative sampling and stochastic gradient descent to calculate the softmax function. These tunable parameters for random walk generation give a advantage over other models by allowing it to model all kinds of networks. However, this model is limited to only considering the topological structure leaving valuable attributes or metadata available with the network.



(a) DeepWalk - Hierarchical Softmax



(b) Random Walk Procedure in Node2Vec

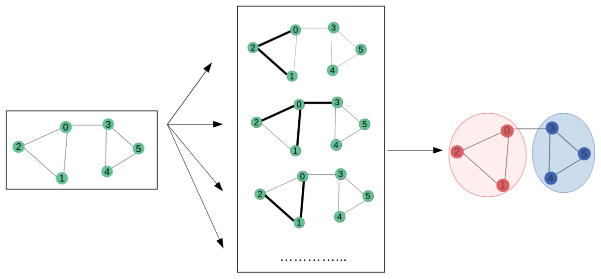
Fig. 2.1 Random Walk Methods

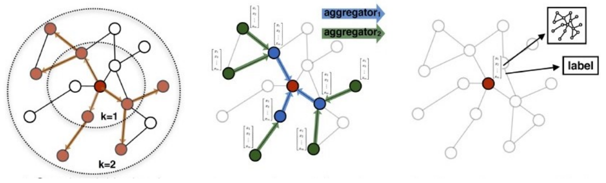
2.2 Neural Network based models

With the significant advancements in neural networks, they are increasingly applied to graph settings. Specifically among them, Convolutional Networks provide a great way to aggregate neighbourhood information using stacked convolutional filters.

Architecture of Graph Convolutional Network (GCN)[3] takes its inspiration from first-order approximations of the spectral convolutions of the graph. Node feature matrix along with adjacency matrix are inputs to the GCN (Fig 2.2(a)). The loss function for GCN consists of graph regularisation loss and supervised loss which is based on the small number of labeled nodes. GraphSAGE[4] is another GNN based on the spatial propagation-based technique. Like GCN, GraphSAGE (Fig

2.2(b)) consider network structure and node features as input. To make the model inductive, GraphSAGE learns aggregation functions at each layer to calculate embeddings using neighbourhood node embeddings. As GraphSAGE is a good inductive model, it can be applied to unseen graphs.


(a) GCN



(b) GraphSage

Fig. 2.2 Neural Network Methods

Though giving very good results, these models suffer from several problems. GCN has problems like suspended animation which means model becomes untrainable after a certain depth and over-smoothing, which prevents deep GCN models, which are necessary to learn complex network structure and attributes. GraphSage has limitations like uniform sampling strategy and not applicability to directed graphs.

2.3 Autoencoder based methods

Autoencoder is a great unsupervised feature learning model. The concept of autoencoder consists of an input layer, k hidden encoding layer, followed by an equal number of hidden decoding layer and an output layer. The dimension of the output is same as input so that the reconstruction loss is used for training the autoencoder weights. DANE[5] is an autoencoder which preserves various proximities present in the network like attribute, first-order and high order proximity. It combines the representations learnt from network topology and attributes effectively as this sources are of very different kinds but simultaneously represent the same nodes. The innermost encoding layer embeddings from attributes and network structure are combined for final representations of nodes.

This chapter gave a brief introduction of some algorithms for NRL tasks. Recent NRL algorithms

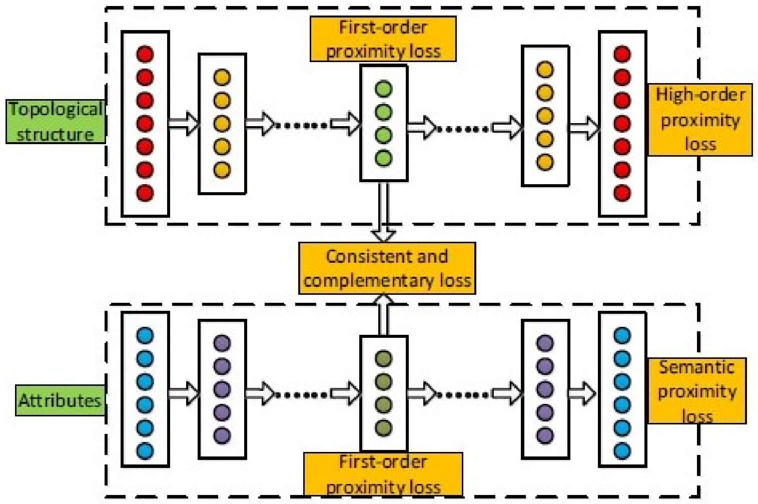

First-order proximity loss
Topological structure
High-order proximity loss
Consistent and complementary loss
Attributes
Semantic proximity loss
First-order proximity loss

Fig. 2.3 DANE Architecture

include node attributes too in the learning process. Most models today have Transformer or GNN as their base model, which consider both network structure and attributes. In the next chapter, we will discuss the formulation of the problem statement.

Chapter 3

Problem Definition

Node attributes are an integral part of the way networks are built. The inclusion of node attributes in the learning process has significantly improved the performance. GCN is an excellent example of that. Using attributes along with the higher hidden layer representation creates the representations that tries to preserve both the original and derived node identities. In this chapter we will try to formulate the problem we are trying to tackle by the means of this project.

3.1 Definitions

Attributed Network: $An\, attributed\, network\, is\, defined\, as\, G\ =\ (V, E, X, Y)\, such\, that\, V\, denotes\, the\, vertices\, present\, in\, the\, network,\ E\, represents\, the\, edges,\ X\, denotes\, the\, node\, attributes,\ and\, Y\, denotes\, the\, node\, labels\, associated\, with\, each\, node\, in\, the\, network,\ i.e\, for\, every\, v_i\, \epsilon\, V,\ there\, exists\, a\ attribute\ vector\ x_i\ and\ y_i\ label\ vector.$

Attributed Network Representation Learning: $Given\, an\, attributed\, network\, G\ =\ (V, E, X, Y),\ represent\ every\ node\ in\ V\ by\ a\ low\ d\ dimensional\ vector\ representation,\ where\ d\ <<\ |V|.$

3.2 Motivation

GraphBert uses node feature attributes of the nodes while learning the node representations as graph residual terms. To learn the node representation, GraphBert does linkless batching. Using the intimacy matrix, top k nodes are sampled for each node, which is used as a subgraph to learn the representation of the node under consideration after passing through several transformer layers. GraphBert uses the raw feature attributes in the residual term and the hidden intermediate representation to learn representation for the next layer. However, GraphBert suffers from one limitation. Subgraph which solely determines the node representations of the node under consideration are entirely constructed by considering the intimacy scores between 2 vertices which are in turn calculated from network structure only. 2 nodes with similar attributes should have close embeddings irrespective of their network structures. However, these node pairs with close attributes which should be considered while learning the representations not considered in the current model.

As GraphBert is a recent model giving excellent results on some well-known datasets, I choose to work on GraphBert. Through this project, I propose to modify the sampling method in the GraphBert by involving attribute vectors as a part of intimacy matrix calculation.

3.3 Problem Statement

Given a graph G = (V,E,X,Y), include the attributes X in the subgraph sampling process of the GraphBert model to improve the results on various downstream tasks.

In this chapter, we looked at the current sampling process and possible extension of the sampling algorithm. At last, we formally defined the problem statement for this project. Next chapters details the proposed solution and experimental results.

Chapter 4

Proposed solution

In this chapter, we will first go through the original GraphBert model followed by explaining in detail what the modified subgraph sampling process stands for.

4.1 GraphBert Working

To overcome the issues like suspended animation and over-smoothing faced by previous GNNs, GraphBert does linkless learning. Along with the issues faced by GNNs, graph sizes today are also very large. So it is not possible to load complete graph in memory. To tackle this problem too, GraphBert's linkless batching helps.

$$S \ = \ \alpha * (I - (1 - \alpha) * \bar{A})^{-1}$$

Intimacy matrix S is calculated as above where $\bar{A} = AD^{-1}$ is normalized adjacency matrix. Here A is the adjacency matrix, D is corresponding diagonal matrix and α is a hyper parameter between 0 to 1. For every node v, top k nodes with highest intimacy score are sampled as subgraph corresponding to node v. For these sampled subgraphs, 4 kinds of node inputs raw feature embedding x_j, weisfeiler-lehman role embedding $WL(v_j)$, intimacy based relative positional embedding $P(v_j)$ and hop based embedding $H(v_j, v_i)$ are given as input in the training process.

$$e_j^x = Embed(x_j)$$
$$e_j^r = Position - Embed(WL(v_j))$$
$$e_j^p = Position - Embed(P(v_j))$$
$$e_j^d = Position - Embed(H(v_j; v_i))$$
$$h_j^0 = aggregate(e_j^x, e_j^r, e_j^p, e_j^d)$$
$$H_i^0 = [h_i^0, h_{i,1}^0, h_{i,2}^0, ..., h_{i,k}^0]^T$$

With this input, GraphBert is pre-trained by considering node attribute reconstruction and graph structure recovery. The propagation equation using attention mechanism are as follows.

$$\begin{aligned} H^l &= G - transformer(H^{l-1}) \\ &= softmax(\frac{QK^T}{\sqrt{d_h}}) + G - Res(H^{l-1}, X) \end{aligned}$$

where d_h is the embedding dimension, G-Res is the graph residual function and,

$$Q = H^{l-1} * W_Q^l$$
$$K = H^{l-1} * W_K^l$$
$$V = H^{l-1} * W_V^l$$

Final representation of the node is calculated as follows.

$$z_i = Fusion(H^D)$$

After this pre-training part, using fully connected layers the model is fine-tuned for the specific downstream task. In my project, I have considered node-classification as the downstream task.

4.2 Modified Sampling

As we clearly see, the intimacy matrix calculation only involves the adjacency matrix from which subgraphs are sampled. To include the attributes in deciding the batched subgraph, we will modify the intimacy matrix calculation as follow.

$$S = \alpha * (I - (1 - \alpha) * A^{'})^{-1},$$
$$where\ A^{'} = \beta * \bar{A} + (1 - \beta) * A^{''}$$

Now $\bar{A}$ is the normalized adjacency matrix and $A^{''}$ is the normalized attribute similarity matrix. $A^{''}$ is calculated as follows.

$$X^{'} = X * X^{T}$$
$$A^{''} = normalized(X^{'})$$

where, X is the attribute vectors matrix. In the above equation, β is another hyper parameter between 0 to 1. Using such combination of adjacency matrix $\bar{A}$ and attribute proximity matrix $A^{''}$, we propose to include the attributes in deciding the subgraphs for the node and ultimately to improve the results on downstream tasks.

4.3 Conclusion

In this chapter, we looked at the working of the GraphBert model and how we modify the subgraph sampling algorithm. In the next chapter, we discuss the extensive experiments done to check the results for the proposed modified sampling.

Chapter 5

Experiments

In the last chapter, we introduced the modified sampling process. In this chapter, we detail the experiments done.

5.1 Datasets and Learning Condition

I experimented with Cora and Citeseer datasets. Description of the datasets is as per following.

Dataset	Nodes	Edges	Attribute Dimension
Cora	2780	5429	1433
Citeseer	3312	4732	3703

Table 5.1 Datasets

In this datasets, less than 5% of the labelled data is used for fine tuning. For the cora dataset, we have pretrained the model for 200 epochs with learning rate for node attribute reconstruction and graph structure recovery both 0.001. For fine tuning part, it is trained for 150 epochs with 0.01 learning rate. For the citeseer dataset, as the original paper authors propose, the convergence may sometime take as long as 2500 epochs at the learning rate of 0.001. To train the model efficiently, I have increased the learning rate to 0.002 and 0.005 in node attribute recovery and graph structure recovery part. To further make training process efficient, I have let it run for 2500 epochs but the learning process is terminated if the training loss is not decreased for 50 epochs. For fine tuning, with learning rate 0.0025, it is trained for 500 epochs.

In all this settings, some default parameters are as follow.

Setting	Value
weight decay rate	$5 * 10^{-4}$
number of attention heads	2
number of hidden layers	2
hidden layer dimension	32
hidden dropout rate	0.5
attention dropout rate	0.3
hidden size	32

Table 5.2 Default settings

In the original paper, they have said to run the fine tuning part several times to get good results. So for each pair of subgraph size $k \in \{1, 2, ..., 9, 10, 15, 20, ..., 50\}$ and $\beta \in \{0, 0.1, 0.2, ..., 0.9, 1\}$, the model was pretrained and fine tuned 10 times separately on cora. For each pair of subgraph size $k \in \{1, 2, ..., 9, 10, 15, 20, ..., 65, 70\}$ and $\beta \in \{0, 0.1, 0.2, ..., 0.9, 1\}$, the model was pretrained and fine tuned 5 times separately on citeseer. Mean, standard deviation and maximum recorded accuracy for each pair of k *and* β is calculated.

5.2 Results

Here we discuss the results of running different existing models on the datasets I have considered.

Model	Cora Accuracy	Citeseer Accuracy
DeepWalk[1]	0.672	0.432
GCN[3]	0.815	0.703
GAT[6]	0.830	0.725

Table 5.3 Different Model on Datasets

5.2.1 Cora

The inclusion of new hyper parameter β improves the performance of the model on the cora dataset. Here we show various graphs to show that. As in figure 5.1, we can see that, for low values of β

like 0 to 0.3, the mean accuracy of 10 measurements is lower. Similarly for high values of β like 0.9 to 1, the accuracy is generally lower.

In this graph,$\beta = 1$ corresponds to the original GraphBert model as for this case, the modified sampling equation becomes same as original one. In the graph 5.2, we plot the mean accuracy of the 10 measurements for original GraphBert. We can clearly see that it increases till k = 8 and then decreases and at last it again increases.

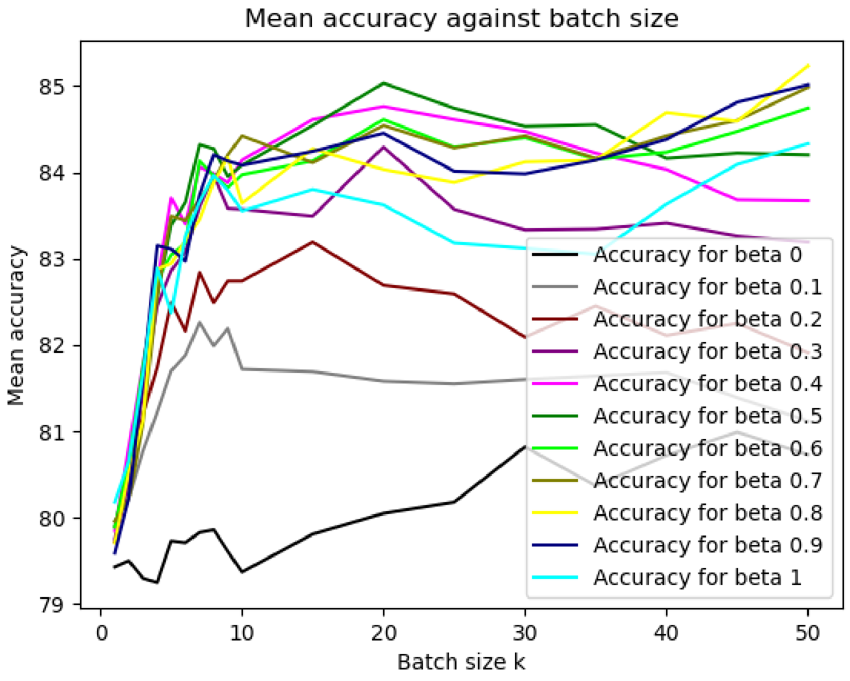


Fig. 5.1 Comparison between k and mean accuracy for different β on Cora

As the table 5.4 shows, after 10 measurements, batch size of 8, 45 and 50 give good results. In table 5.5, measurements for $\beta = 0.5$ are given. From comparing the 2 tables, we can see that, for almost all values of k, modified sampling GraphBert beats the original one. For k = 20, their is sharp increase in mean accuracy and maximum observed accuracy and there is also decrease in the standard deviation. Also another general observation is that, as beta increases from 0 to 0.6 to 0.8, the accuracy improves and decreases as beta increases further.

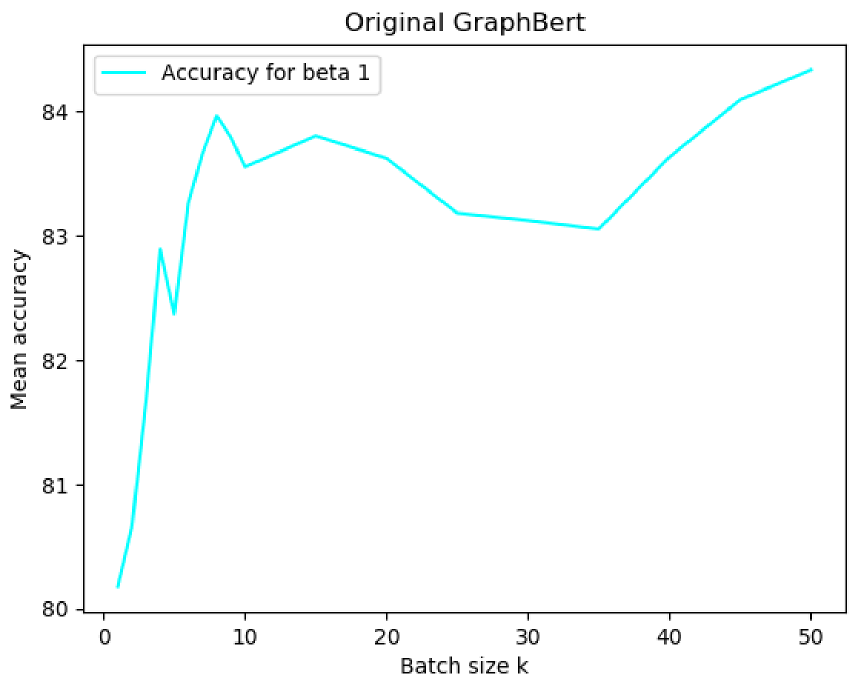


Fig. 5.2 Original GraphBert on Cora

Batch size	Mean accuracy	Standard deviation	Maximum accuracy
1	80.18	0.52	81.2
2	80.65	0.42	81.3
3	81.66	0.41	82.4
4	82.89	0.58	84.0
5	82.37	0.31	82.9
6	83.26	0.48	84.0
7	83.66	0.38	84.3
8	83.96	0.41	84.7
9	83.79	0.27	84.5
10	83.55	0.38	84.5
15	83.8	0.54	84.7
20	83.62	0.36	84.3
25	83.18	0.55	84.0
30	83.12	0.28	83.5
35	83.05	0.14	83.2
40	83.63	0.55	84.9
45	84.09	0.55	85.0
50	84.33	0.41	85.0

Table 5.4 Statistics for Original GraphBert on Cora

Batch size	Mean accuracy	Standard deviation	Maximum accuracy
1	79.9	0.49	80.9
2	80.57	0.45	81.0
3	81.69	0.31	82.3
4	82.56	0.38	83.2
5	83.39	0.42	84.0
6	83.65	0.41	84.4
7	84.32	0.45	85.1
8	84.27	0.28	84.7
9	83.95	0.46	84.6
10	84.08	0.42	84.9
15	84.54	0.35	85.2
20	85.03	0.32	85.7
25	84.74	0.31	85.2
30	84.53	0.39	85.2
35	84.55	0.3	85.0
40	84.16	0.39	84.8
45	84.22	0.46	85.1
50	84.2	0.35	84.8

Table 5.5 Statistics for modified GraphBert on Cora for beta = 0.5

5.2.2 Citeseer

We apply the same idea on citeseer dataset. But from the observations, I got somewhat unexpected result. For all values of k and beta, the mean accuracy and maximum observed accuracy differ by maximum of 0.5 as it can be seen from the fig 5.3. The results got from citeseer dataset are not very clear to deduce any kinds of trends. Table 5.6 and 5.7 shows the results got are very similar for different pairs of k and β.

In the next section, we try to justify the possible reasons for such observations on citeseer dataset.

5.3 Reasoning for results on Citeseer

By introducing a new hyper parameter β, we are basically superimposing 2 networks, one constructed from the network edges and other constructed from the attribute similarity. Imposing 2 networks on each other makes the graph dense. But if the number of edges provided by one network

Batch size	Mean accuracy	Standard deviation	Maximum accuracy
1	70.42	0.31	70.8
2	70.08	0.19	70.4
3	70.02	0.23	70.3
4	70.12	0.19	70.3
5	70.18	0.29	70.7
6	70.36	0.21	70.6
7	70.0	0.28	70.4
8	70.22	0.15	70.4
9	69.9	0.24	70.3
10	70.06	0.17	70.3
15	70.12	0.21	70.4
20	70.1	0.17	70.3
25	70.16	0.2	70.5
30	70.24	0.22	70.5
35	69.96	0.17	70.2
40	69.94	0.1	70.1
45	69.98	0.32	70.5
50	69.94	0.16	70.1
60	70.16	0.19	70.5
70	70.02	0.26	70.4

Table 5.6 Statistics for Original GraphBert on Citeseer

Batch size	Mean accuracy	Standard deviation	Maximum accuracy
1	70.48	0.6	71.2
2	70.38	0.16	70.5
3	70.1	0.28	70.5
4	70.36	0.14	70.5
5	70.06	0.19	70.2
6	69.92	0.15	70.2
7	69.94	0.19	70.3
8	69.98	0.22	70.3
9	69.84	0.42	70.4
10	70.06	0.08	70.2
15	70.28	0.16	70.5
20	70.2	0.4	70.7
25	70.04	0.29	70.5
30	70.12	0.26	70.6
35	70.1	0.09	70.2
40	69.98	0.31	70.4
45	70.04	0.29	70.4
50	70.14	0.3	70.5
60	70.38	0.31	70.8
70	69.66	0.46	70.3

Table 5.7 Statistics for modified GraphBert on citeseer for beta = 0.5

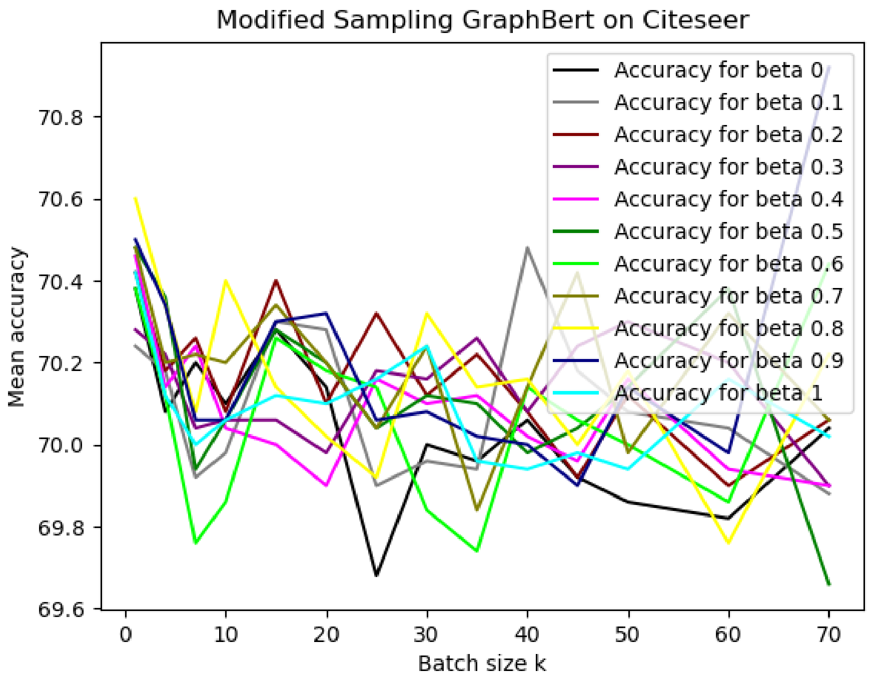


Fig. 5.3 Comparison between k and mean accuracy for different β on Citeseer

are too few compared to the edges provided by other network, the overall impact of the 1st network is negligible. In case of citeseer dataset, number of edges present in the attribute network are more than 3900000, while number of edges in network structure are less than 5000. There are merely 200 edges in original network which are not present in attribute constructed network (We say 2 nodes are connected in attribute network if their cosine similarity is non zero). In cora dataset, the number of edges in attribute network are much lesser and distinct edges in network structure are more compared to number of nodes. I believe this is the reason behind the surprising result on citeseer dataset.

Chapter 6

Discussion and Conclusion

As observed in the previous chapters, we propose to modify the subgraph sampling algorithm of GraphBert. On testing the proposed solution on 2 standard graph datasets, improvement in results is observed in 1 dataset, while in another, almost equivalent results were produced. I have tried to justify the results to the best of my knowledge. In the future work, instead of superimposing 2 networks, we can study how information would propagate in the attribute constructed network and original network and after learning separate representations, how those 2 representations of the same node from different contexts can be combined.

References

[1] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk,” *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, Aug 2014.

[2] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” 2016.

[3] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” 2017.

[4] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” 2018.

[5] H. Gao and H. Huang, “Deep attributed network embedding,” in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 3364–3370, International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[6] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” 2017.