Deep Learning on Graphs with Graph Convolutional Network

Himanshu Chauhan

Computer Science Department, California State University Sacramento

himanshuchauhan@csus.edu

***Abstract*— Machine learning on graphs has become an important and ubiquitous task with a wide range of applications. With an advancement in the field of artificial intelligence and machine learning, analysis and research on graphs have been receiving more and more attention because of the great expressive power of the graph. The motivation for graph neural network goes back to convolutional neural network. CNNs have the ability to capture spatial information and compose them to construct highly expressive representations, which led to breakthroughs in almost all machine learning areas and started the new era of deep learning. However, CNN's can only operate on regular Euclidean data like images and text while these data structures can be regarded as instances of graphs. Also, unlike standard neural networks, graph neural networks retain a state that can represent information from its neighbourhood with arbitrary depth. This project aims to provide an introduction to graph convolutional network using Deep Graph Library, a python package dedicated to deep learning on graphs. I present a series of tutorials which I developed to help anyone new to Graph Convolutional Network (GCN). The tutorials explain the following concepts in detail: introduction to pytorch, followed by creation of simple graphs using DGL and model implementation and finally implementation of GCN using Cora dataset.**

Keywords — graph convolutional network, deep graph library, node classification, semi-supervised learning, PyTorch, graph-structured data

I. INTRODUCTION

If we have lot of data in the form of a graph, how will we get this kind of graph structure into some kind of deep learning architecture? Today, a number of learning tasks require dealing with graph data which contains rich relation information among elements. Numerous important real-world data sets come in the form of the graphs or networks e.g. social networks, knowledge graphs, worldwide web, protein interaction network etc. Modelling physics system, learning molecular fingerprints, predicting protein interface, and classifying disease requires that the model learns from graph inputs.

GCNs are a very powerful neural network architecture for machine learning on graphs. In fact, they are so powerful that even a randomly initiated 2-layer GCN can produce useful feature representations of nodes in networks. A graph convolutional network (GCN) is a neural network that operates on graphs. Given a graph G = (V, E), a GCN takes as input:

• An input feature matrix N × F^0 feature matrix, X, where N is the number of nodes and F^0 is the number of input features for each node, and

• An N × N matrix representation of the graph structure such as the adjacency matrix A of G. [1]

For example, the figure below illustrates a 2-dimensional representation of each node in a network produced by such a GCN. [1]

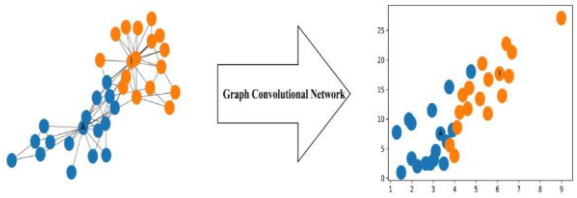

Fig-1 2D Representation of a graph produced by GCN

Despite the increasing attention given to the graphs and their implementation with the machine learning concepts, there are still not many resources available explaining these concepts in detail. It is tough to find detailed consolidated information on implementation of GCN especially for a beginner at one place. By means of this project my goal is to present a step by step implementation of graph convolutional network from scratch with detailed explanation of the code.

Since, I am a beginner in the area of GCN it was challenging to find the resources to understand the underlying concepts of GCN. The approach I took was to first gather as many resources as possible which were available online and went through them to extract the information on GCN concepts. With the help of these resources I implemented and trained a GCN model using deep graph library (DGL). DGL is built on top of deep learning frameworks such as pytorch (an open source library for deep learning platform developed by Facebook) and Apache MXNET. As DGL is supported by pytorch, it was necessary for me to learn the basic concepts of pytorch and gain hands on experience with the implementation of the machine learning models using pytorch. I have implemented a semi-supervised learning technique for classifying nodes in a graph using GCN.

There are not many good tutorials to guide a beginner like me, to learn GCN implementation using pytorch from scratch. Therefore, I developed a series of 6 tutorials to facilitate any new learner in this field, listed as follows:

1. Introduction to pytorch
2. Classification model implementation using pytorch
3. CNN Implementation using pytorch
4. Graph creation using DGL
5. Graph Neural Network in DGL for node classification
6. Graph Convolutional Model implementation in semi-supervised learning for node classification

The following sections would explain in detail the work, experiments, results and analysis based on the tutorials mentioned above.

II. PROBLEM FORMULATION

2.1 Semi-Supervised node classification

In a semi-supervised learning classification problem, a large amount of data is unlabelled and together with the labelled dataset a model is trained to build a classifier.

- A graph consists of nodes, labels, edges and features.
- The model is trained by giving the graphs and features as inputs.
- The model will predict the class for each node in the graph. Since this is a semi-supervised learning, a large number of nodes in the graph are unlabelled.
- The loss for training the model is calculated on the labelled nodes in the graph. The model is trained to reduce the loss with respect to the labelled part of the graph.
- Figure-5 represents Schematic depiction of multi-layer Graph Convolutional Network (GCN) for semi-supervised learning with C input channels and F feature maps in the output layer. The graph structure (edges shown as black lines) is shared over layers, labels are denoted by Yi. [5]

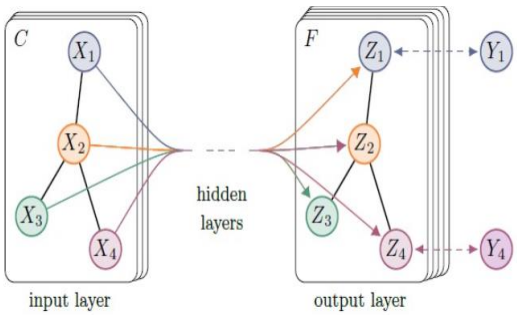

Fig-2 Graph Convolutional Network

2.2 Datasets

The datasets used are:

2.2.1 Zachary's Karate Club Dataset

- Total nodes: 34
- Labelled nodes: 2 (node 0 and node 33)
- Input given: graph and features (one-hot encoded)
- Calculated loss on labelled nodes to train model
- Output: Predicted label for unlabelled nodes

2.2.2 Cora Graph Dataset

- Total nodes: 2708
- Labelled nodes: 140 (5% of total nodes)
- Total edges: 10556
- Total Features: 1433 features for each node
- Input given: graph and features
- Calculated loss on labelled nodes to train model
- Output: Predicted label for unlabelled nodes

III. SYSTEM / ALGORITHM DESIGN

3.1 Pytorch Tutorials

The tutorials mentioned in this section will provide a good understanding and concepts required for implementing machine learning models using pytorch library. It covers the following concepts.

- Basic Introduction to Pytorch and tensors.
- Computational Graphs in pytorch.
- Pytorch Autograd
- The forward and backward function commonly seen in NNs model in Pytorch.
- Fully Connected Neural Network model implementation for binary and multi-class classifier using pytorch. Data set used was network intrusion dataset. The model predicted good or bad connections for binary classifier while the type of attack for multi-classifier.
- Also used titanic data set for implementing a classification model using pytorch to predict whether the passenger survived or not.
- Implemented Convolutional Neural Network (CNN) model using pytorch. The data set used was CIFAR10 for image classification.

3.2 Graph Neural Network using DGL for node classification

The tutorials mentioned in this section will provide an overview of DGL library and understand how it enables computation on graphs from a high level. It covers the following concepts in detail.

- Graph creation using DGL
- Create and add nodes and edges in a graph
- Assigning features to nodes and edges in the graph
- Multi-graph basics

- Useful deep learning graph functions
  - Message passing operation for updating node features which includes sending and receiving message by the nodes.
  - Started with building a small sized well-known graph named “Zachary’s Karate Club” for the 1st basic GCN model implementation, shown in figure-2.
  - Step by step guide to build the below graph (figure-3) from scratch, that is, from creation of nodes to assigning the features to the nodes. The number of nodes in the graph are 34 and the input features are given as one-hot encoded vectors.
  - The karate club graph dataset is a social network which captures 34 members and documents pairwise links between members who interact outside the club. The club later divides into two communities led by the instructor (node 0) and the club president (node 33).[2]
  - The model was implemented for classifying nodes that is the instructor and club president.

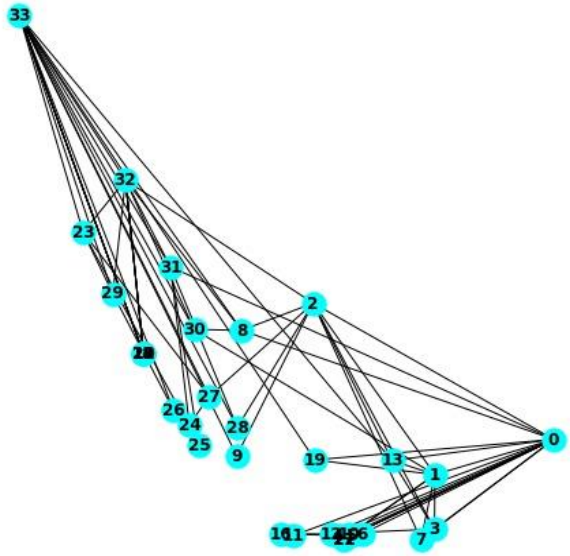

Fig-3 Zachary’s Karate Club graph

3.3 GCN Model Implementation

This tutorial is built on top of earlier tutorials. It is imperative to understand the previous tutorials before starting this one, as those tutorials are the building blocks for the current tutorial.

The tutorials mentioned in this section will provide a gentle introduction and implementation of semi-supervised classification with Graph Convolutional Network (GCN) using deep graph library and demonstrate how DGL combines graph with deep neural network and learn structural representation. [3]

- Covers basic uses of DGL API.
- Uses Vanilla GCN model consisting of two GCN layers
- GCN from the perspective of message passing, that is, defining the message and reduced function. The GCN layer essentially performs message passing on all the nodes to perform message propagation throughout the graph.
- Uses CORA dataset for training the model (figure-4). The Cora dataset consists of 2708 scientific publications (that is 2708 nodes) classified into one of seven classes (that is 7 classes in the graph). Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.[4]

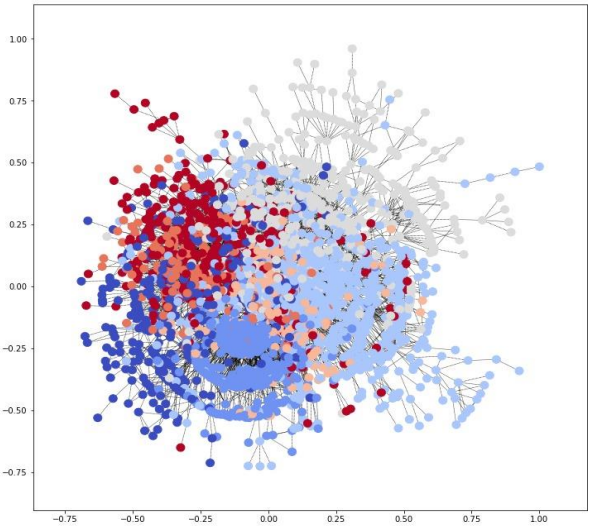

Fig-4 2D representation of Cora Graph dataset

- The input feature size is 1433 and the total number of classes are 7. Figure-5 shows the count for each label in the graph.

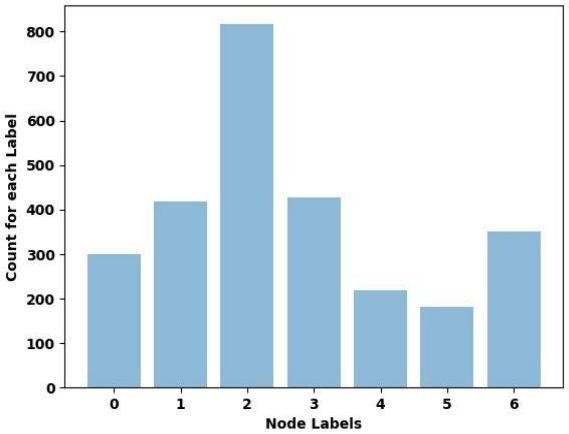

Fig-5 Count for each class label (7) in Cora graph

- Since this is a semi-supervised classification problem, only few nodes would be labelled while other nodes will not have any label.
- The model predicts the class for unlabelled node.

IV. EXPERIMENTAL EVALUATION

4.1 Zachary's Karate Club Dataset

- Total nodes: 34
- Labelled nodes: 2 (node 0 and node 33)
- Input given: graph and features (one-hot encoded)
- Calculated loss on labelled nodes to train model
- Output: Predicted label for unlabelled nodes

4.1.1 Experimental Settings

- Two GCN layers
- Input Size: 34; Hidden size: 5; Output size: 2
- Optimizer: Adam
- Learning Rate: le-3
- Activation function: relu
- Output Layer Activation function: SoftMax
- Loss Function: nll_loss
- Used NetworkX python package for studying and analysing the graph structure
- Number of Epochs: 40

4.1.2 Results

This dataset was very small and only had 2 labelled nodes. So, the accuracy for this model was high and loss was very low (0.0001).

Figure-6 shows the visualization of the graph during the training process for epoch-0, by plotting the output feature in a 2D space. In this epoch the model predicted the same label, that is 1 for all the nodes. Therefore, the colour of all the nodes is same.

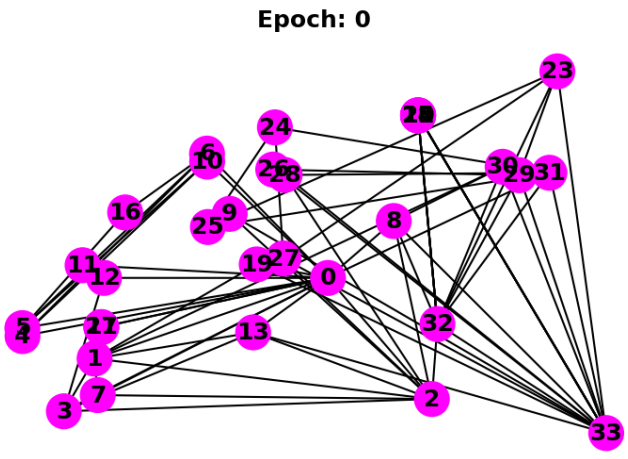

Fig-6 Epoch: 0

Figure-7 shows the visualization of the graph during the training process for epoch-39, by plotting the output feature in a 2D space.

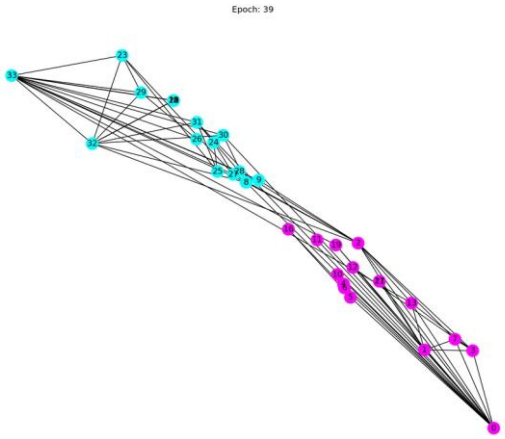

Fig-7 Epoch: 39

I have also made an animation of all the epochs (40) during the training process.

4.2 Cora Graph Dataset

- Total nodes: 2708
- Labelled nodes: 140 (5% of total nodes)
- Total edges: 10556
- Total Features: 1433 features for each node
- Input given: graph and features
- Calculated loss on labelled nodes to train model
- Output: Predicted label for unlabelled nodes
- Trained on 140 labelled nodes
- Tested on 300 and 1000 unlabelled nodes on the same graph dataset

4.2.1 Experimental Settings

- 2,3 GCN layers
- 1,2 dropout layers
- Input Size: 1433; Hidden size: 32, 16; Output size: 7
- Optimizer: Adam
- Learning Rate: le-3
- Activation function: relu
- Output Layer Activation function: softmax
- Loss Function: nll_loss
- Used NetworkX python package for studying and analysing the graph structure
- Number of Epochs: 300,500,1000,1500
- Metrics Used for model evaluation: Accuracy Score, F1 Score, Recall Score, Precision Score

4.2.1 Results

Below is the table for all the matrices score for different settings and number of nodes used for testing used for model evaluation.

TABLE-2
Results of GCN model

Model	Accuracy	F1 Score	Recall	Precision
2 GCN (500 epochs) **Train** data 140 nodes	0.97	0.9722	0.977	0.977
2 GCN (500 epochs) **Test** data	0.661	0.6488	0.6388	0.6278
3GCN-layers (1500 epochs) + 2 Dropout Layers **Train** data 140 nodes	**0**.89	0.88	0.89	0.87
3GCN-layers (1500 epochs) + 2 Dropout Layers **Test** data 1000 nodes	0.65	0.64	0.65	0.64
3GCN-layers (1500 epochs) + 2 Dropout Layers **Test** data 300 nodes	0.73	0.73	0.723	0.73
3GCN-layers (1000 epochs) + 1 Dropout Layer **Train** data 140 nodes	0.98	0.98	0.98	0.98
3GCN-layers (1000 epochs) + 1 Dropout Layer **Test** data 1000 nodes	0.76	0.76	0.74	0.74
3GCN-layers (1000 epochs) + 1 Dropout Layer **Test** data	0.7833	0.76	0.76	0.78

In the earlier experiments the model performed very well on the training data but it did not perform very well on the test data probably due to overfitting. I added dropout layers and the performance of the model was improved by about 15 %

Figure-8 shows the plotting of Confusion Matrix for train (140 labelled nodes) and test data (300 nodes)

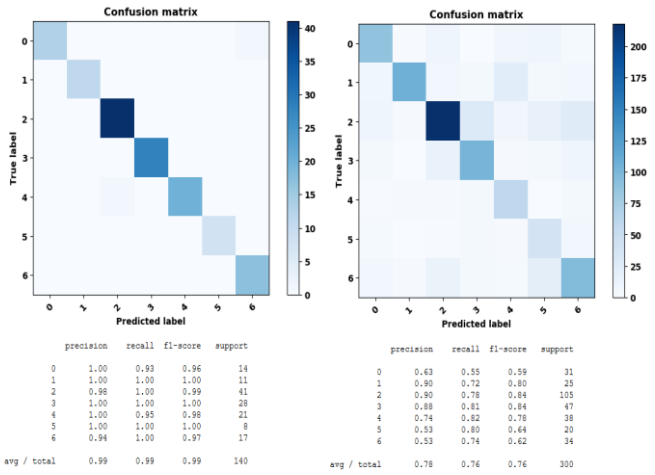

Fig- 8 Confusion Matrix for train(left) and test nodes(right)

Figure-9 and Figure-10 shows the visualization of the graph after the training process for epoch 0 and epoch 1000 respectively, by plotting the output feature in a 2D space. I have used T-distributed Stochastic Neighbor Embedding (t-SNE) algorithm to embed the output after training the model into 2D Space. We can see how the embeddings for nodes in the graph has changed from epoch-0 to epoch-1000.

t-SNE is a machine learning algorithm for visualization developed by Laurens van der Maaten and Geoffrey Hinton. It is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modelled by nearby points and dissimilar objects are modelled by distant points with high probability. [6]

Graph embeddings are the transformation of graphs to a vector or a set of vectors. Embedding should capture the graph topology, vertex-to-vertex relationship, and other relevant information about graphs, subgraphs, and vertices. [7] Graph embeddings have become very popular with very large size graph data which helps to preserve the features of the graph. There are many embedding techniques available which can be used for node and graph embeddings. I have not used any graph embedding on the input features, since the dataset I

am using is not very large. However, I have used t-SNE algorithm to embed the output into 2D space for visualization.

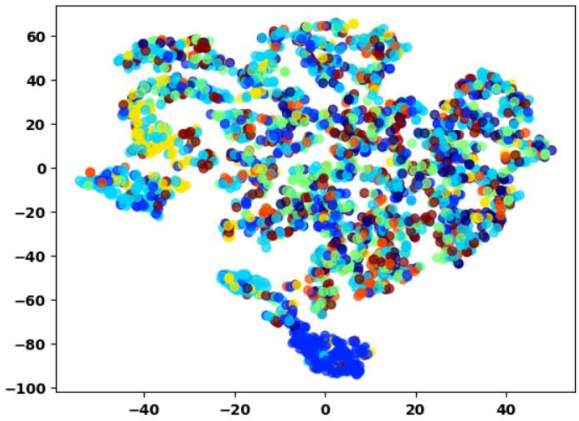

Fig-9 2D representation of the graph using t-SNE for epoch-0

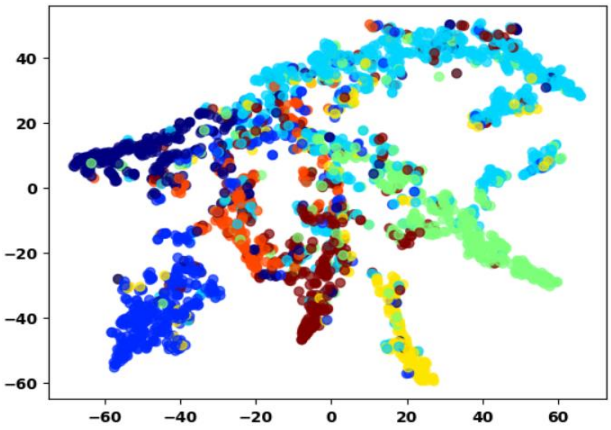

Fig-10 2D representation of the graph using t-SNE for epoch-1000

Figure-11 shows the visualization of the graph after the training process, by plotting the output feature in a 3D space. I have used the t-SNE algorithm to embed the output after training the model into 3D Space.

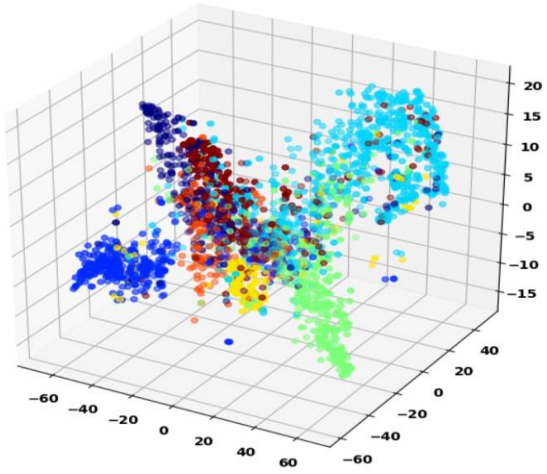

Fig-11 3D representation of the graph using t-SNE for epoch-1000

V. Related Work

There are numerous papers published with the similar kind of implementation of the GCN model. There are some online resource available too. But, as the field is still evolving, all the content related to the topic being presented in this paper is segregated all over the place. It is extremely difficult and cumbersome for anyone new to field to understand the GCN model implementation. It is not just about the model implementation but one must also know the concepts of the neural networks and implementation using pytorch.

The tutorial presented in this paper, serves as a step by step guide to ramp-up on pytorch, gradually learn and understand the GCN model implementation using DGL and pytorch. The tutorials being presented are exclusive and comprehensive, which helps the user to sequentially learn the necessary skills before directly jumping to GCN implementation details. These tutorials make the life easy, especially for the beginners who have no prior pytorch or DGL library skills.

VI. Conclusion

In this project I have created means of learning pytorch, DGL library and GCN model implementation systematically. The tutorials presented in this paper serve as comprehensive guide to gain the necessary knowledge needed to implement GCN model. I have implemented a semi-supervised classification with graph convolutional network for node classification using a moderately sized dataset named CORA.

VII. Work Division

Worked independently in this project.

VIII. LEARNING EXPERIENCE

- Learnt pytorch basics
- Implemented classification Neural network model using pytorch
- Learnt DGL library and its implementation
- Implemented and understood semi-supervised classification with graph convolutional network
- Learnt about other libraries like NetworkX
- Understood how to create graphs from scratch, assign features to nodes and edges, and other basic functions on graphs.
- Understood and implemented message passing techniques used by nodes in the graph to update its features with the information sent from neighboring nodes.
- Learnt to implement GCN model using pytorch library

- The project was totally new and I had no prior experience in pytorch or GCN. Working on this project was challenging and a great learning opportunity.
- It helped me in understanding in what way DGL aids computation on graph and train a graph neural network in DGL to categorize nodes in a graph.
- I have also learnt the t-SNE algorithm and its importance in implementation of node embeddings for dimensionality reduction.

REFERENCES

[1] https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graphconvolutional-networks-7d2250723780
[2] https://docs.dgl.ai/tutorials/basics/1_first.html
[3] https://docs.dgl.ai/tutorials/models/1_gnn/1_gcn.html
[4] https://relational.fit.cvut.cz/dataset/CORA
[5] https://openreview.net/pdf?id=SJU4ayYgl
[6] https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding
[7] https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007