**(Q1) Increase the N value from 20 (original value) to 200 with multiple N values in between and observe the change of graph density and degree distribution (i.e., histogram plot)**

After increasing N to 200 and E to 400, the following was observed:

**Graph Density:** The graph density was much lower than the original graph. This is because the number of possible edges in a graph increases quadratically with the number of nodes, while we only doubled the number of edges.

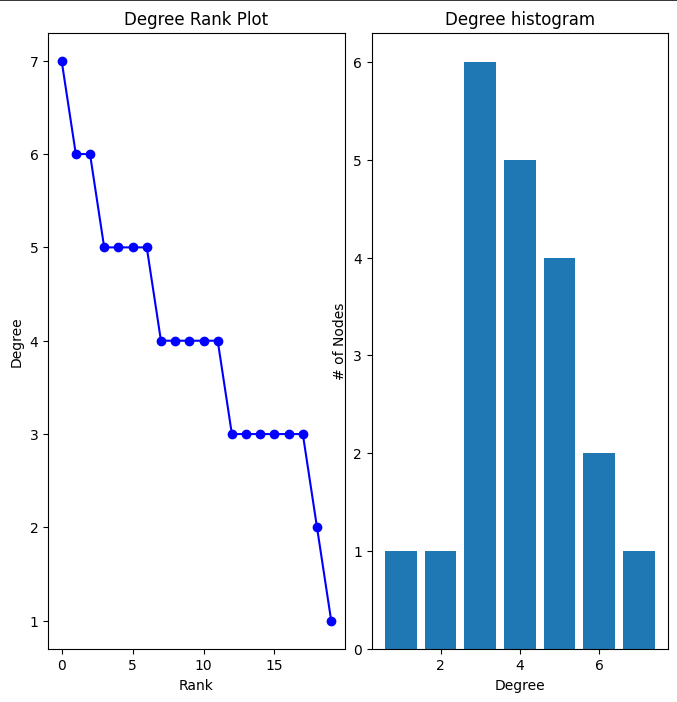
**Degree Distribution:** The degree distribution was more concentrated around the average degree. In the original graph with 20 nodes, there was a wider spread of degrees. With 200 nodes, the degree distribution looked more like a normal distribution, with most nodes having a degree close to the average. This is a characteristic of Erdős-Rényi random graphs as the number of nodes increases.

This behavior is related to the law of large numbers: as the graph size increases, the actual number of edges connected to a node will converge towards its expected value, which is determined by the probability of edge creation in the Erdős-Rényi model.

Attached below are two snapshots.

*N=20, E=40 and graph density= 0.21052631578947367 (left image)*

*N=200, E=400 and graph density=0.020100502512562814 (right image)*

 A graph of a degree and histogram

Description automatically generated

**(Q2)**

1. **Explain the differences between supervised learning, self-supervised learning, and semi-supervised learning methods.**

**Supervised Learning:** Uses labeled data for training. Each input has a known output label. Common in classification and regression tasks but requires a lot of labeled data.

**Self-Supervised Learning:** A machine learning technique that uses unlabeled data to train models to create their own labels. The model generates its own supervision by predicting parts of the data from other parts. Often used in pre-training models like language models.

**Semi-Supervised Learning:** Combines a small amount of labeled data with a large amount of unlabeled data. It helps when labeled data is scarce but unlabeled data is abundant.

1. **Explain the differences between transductive learning and inductive learning.**

**Inductive Learning:** Generalizes from specific training data to make predictions on unseen, new data. The model learns a general rule during training and applies it to any future data.

**Transductive Learning:** Focuses on predicting specific outputs for the given test data directly without generalizing to unseen data. It learns from both training and test data together to make predictions for the test set only.

The key difference is that inductive learning aims to generalize to unseen data, while transductive learning is restricted to a specific set of test instances.

**(Q3)**

1. **Increase the number of epochs from 50 to 500 and observe the change in validation accuracy.**

Initially with 50 epochs, the model had a training accuracy of 100% but a validation accuracy of only 64.71% with a loss of 0.7593. The results of the 40th epoch is illustrated in the snapshot below.

A screen shot of a graph

Description automatically generated

However, when trained with 500 epochs, the model had a training accuracy of 100% and a validation accuracy of 82.35% with a loss of only 0.018. The results of the 490th epoch is illustrated in the snapshot below.

A graph showing a number of data

Description automatically generated with medium confidence

1. **Experiment without self-loops added to GCNConv() layers in the GCN() model and detail the model accuracy increase/decrease.**

When experimented without self-loops in the GCNConv layers by adding the parameter *add\_self\_loops=False* and training with 50 epochs, it was observed that both the training and validation accuracy reduced to 50% and 47.06% respectively with a loss of 0.8575. Attached below is a snapshot of the 40th epoch.

**A graph of a graph showing a number of dots

Description automatically generated with medium confidence**

1. **Increase the number of GCNConv() layers in the GCN() model upto 8 layers from original 3 layers. Detail the accuracy increase/decrease.**

Increased the layers to 8 with 2 input features and 2 output features per layer from the 4th layer to the 8th layer. After training for 50 epochs, it was observed that the training accuracy was 50% and validation accuracy was 20.59% with a loss of 1.2485. Attached below is a snapshot of the 40th epoch.

A graph of a graph with numbers and dots

Description automatically generated with medium confidence

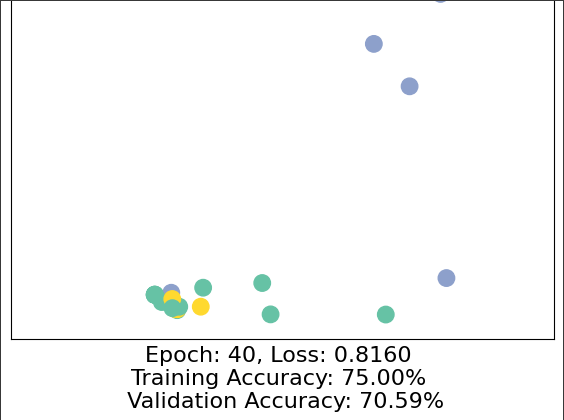
1. **Using best performing values for *in\_channels* and *out\_channels* in *GCNConv()***

Used the values as illustrated in the snapshot below.

A screen shot of a computer

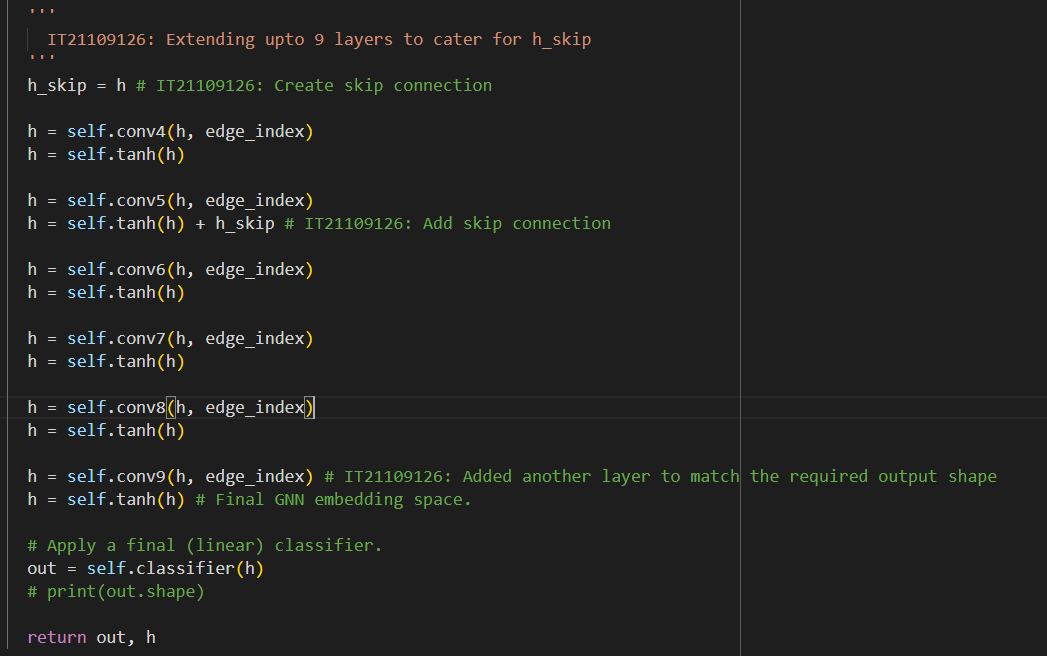
Description automatically generated

After training for 50 epochs, it was observed that there was a training accuracy of 75% and a validation accuracy of 70.59% with a loss of 0.8160. Illustrated below is a snapshot of the 40th epoch.



1. **Add skip connections between some of the GCNConv() layers.**

After creating a skip connection and extending to 9 layers from 8 layers to match the required output shape, the training accuracy remained the same at 75% but the validation accuracy dropped to 41.18% with a loss of 0.8402. The below snapshots illustrate the modified code and the results of the 40th epoch.



A screen shot of a graph

Description automatically generated

**(Q4) Explain the differences between Message Passing GNN, graph convolution network (GCN), graph attention network (GAT) and GraphSAGE.**

**Message Passing GNN (MP-GNN):** A general framework where nodes iteratively update their representations by aggregating features from their neighbors through multiple message-passing steps.

**GCN (Graph Convolutional Network):** Aggregates node features using a normalized sum or mean of neighbors' features. Treats all neighbors equally, making it simple but lacking the ability to assign importance to specific neighbors.

**GAT (Graph Attention Network):** Introduces attention mechanisms to assign different weights to neighbors during aggregation, allowing the model to focus on more important nodes dynamically, leading to better performance on heterogeneous graphs.

**GraphSAGE:** Samples a fixed number of neighbors and applies various aggregation techniques (mean, LSTM, pooling) to generate node embeddings. Designed for inductive learning, meaning it can generalize to unseen nodes and is more scalable for large graphs.

**Github Repo Link:** <https://github.com/it21109126/DL-Lab-6>