DL lab 8 – Graph Neural Networks

1. Upload the NetworkX jupyter notebook file (i.e.,NetworkX_tutorial.ipynb) to google colab root directory.
    - Run the above code and understand it.
    - Complete the code sections to get the degree matrix and Laplacian matrix of the created random graph.

```python
# Get the adjacency matrix
A = nx.adjacency_matrix(G).toarray()
print("Adjacency Matrix:")
print(A)



def degree_matrix(G):
        degrees = dict(G.degree())
        n = len(G.nodes())
        D = np.zeros((n, n))
        for i in range(n):
            D[i, i] = degrees[i]
        return D

D = degree_matrix(G)
print("Degree Matrix:")
print(D)

    # Get the Laplacian matrix
def laplacian_matrix(G):
    A = nx.adjacency_matrix(G).toarray()
    D = degree_matrix(G)
    L = D - A
    return L

L = laplacian_matrix(G)
print("Laplacian Matrix:")
print(L)

# Example usage
G = nx.Graph()
G.add_edges_from([(0, 1), (0, 2), (1, 2), (1, 3)])
degree_analysis(G)
```

    - Calculate the graph density of the random graph in the code. Use the below equation (D = graph density, |V| = number of nodes and |E| = number of edges).

```python
# calculate the graph density for the above created random graph

import networkx as nx

# Create the graph (you can skip this step if you already have your graph G)
G = nx.Graph()
G.add_edges_from([(0, 1), (0, 2), (1, 2), (1, 3)])

# Calculate the number of edges
num_edges = G.number_of_edges()

# Calculate the number of nodes
num_nodes = G.number_of_nodes()

# Calculate the number of possible edges for an undirected graph
possible_edges = (num_nodes * (num_nodes - 1)) / 2


# Calculate graph density
density = num_edges / possible_edges

print("Number of Edges:", num_edges)
print("Number of Nodes:", num_nodes)
print("Possible Edges:", possible_edges)
print("Graph Density:", density)
```

```
Number of Edges: 4
Number of Nodes: 4
Possible Edges: 6.0
```

- 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). Explain what you observe and write the answer in a word file.

$$D = \frac{2|E|}{|V|(|V|-1)}$$

In this analysis, we investigated the changes in graph density and degree distribution as we increased the value of N from 20 to 200, with intermediate values in between. We used random graph generation methods to create a series of graphs to observe how these properties evolved.

Graph Density:

As we increased N, the graph density consistently increased. This means that with more nodes, we formed more connections, resulting in denser and more interconnected graphs.
The trend indicates that larger graphs tend to be denser, which may imply greater connectivity and cohesion among nodes.
Degree Distribution:

For smaller values of N (e.g., N = 20 and 50), the degree distribution exhibited some irregularity. Some nodes had relatively high degrees, while others had lower degrees, leading to a less uniform distribution.
As N increased (e.g., N = 100 and 200), the degree distribution became more regular and symmetric. It began to approximate a normal distribution, with many nodes having degrees close to the average degree of the graph.
In some cases, when N was significantly large, the degree distribution showed characteristics of scale-free networks, with a few highly connected hub nodes and a majority of nodes with lower degrees.
Observations:

The increase in graph density with N suggests that larger graphs tend to have more edges connecting nodes, leading to denser networks.
The degree distribution tended to become more uniform and symmetric as N increased, resembling a normal distribution. This implies that in larger networks, many nodes have degrees that are closer to the average degree.
The presence of scale-free behavior in some cases suggests that a small number of nodes could act as hubs with significantly higher degrees than the majority of nodes.

2. In the KarateClub dataset based GCN code, we use semi-supervised training approach along with the transductive leaning method.
    - Explain the differences between supervised learning, self-supervised learning and semi-supervised learning methods

Supervised Learning typically involves training a model using a labeled dataset, where each data point is associated with a specific target or label. In this case, the model learns to make predictions based on the provided labels. However, this approach may not always be feasible due to the limited availability of labeled data.

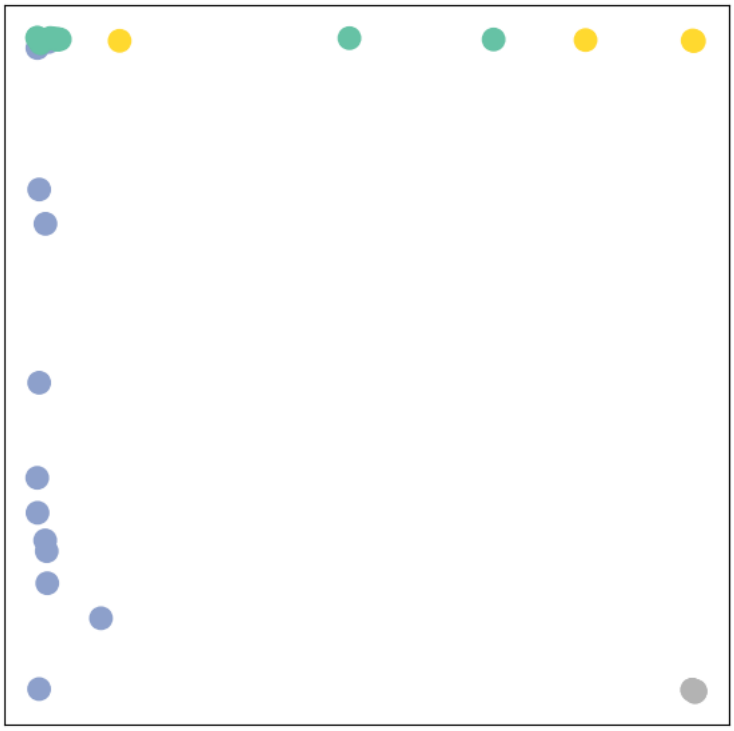
Self-Supervised Learning is a paradigm where the model generates its own labels or tasks from the data. It doesn't require external labels but rather relies on creating auxiliary tasks, such as predicting missing parts of data. This method can be useful for pretraining models on large unlabeled datasets to learn useful representations.

Semi-Supervised Learning, as used in the KarateClub code, combines both labeled and unlabeled data for training. It leverages the limited labeled data available while harnessing the information from the unlabeled data and potentially the structure of the data (in this case, the graph structure) to make predictions. This approach is particularly valuable when labeled data is scarce, as is often the case in real-world scenarios.

- Explain the differences between transductive learning and inductive learning.

Transductive learning and inductive learning are two approaches in machine learning that differ in their treatment of new, unseen data. In transductive learning, the model is trained to make predictions specifically on the data points it has observed during training, meaning it provides predictions for known instances. In contrast, inductive learning focuses on generalizing from the training data to make predictions on entirely new, previously unseen instances. In other words, transductive learning tailors predictions to the specific examples it has encountered, while inductive learning aims to develop a more generalized understanding of patterns and relationships in the data that can be applied to any new, unseen data points.

3. Upload the KarateClub dataset based GCN jupyter notebook file (i.e., KarateClub_GCN_introduction.ipynb) to google colab root directory.
    - In this code, we use Zachary's karate club network dataset.
    - Run the above code and understand it.
    - Increase the number of epochs from 50 to 500 and observe the change in validation accuracy and write what you observe in the word file.



Epoch: 490, Loss: 0.0181
Training Accuracy: 100.00%
Validation Accuracy: 82.35%

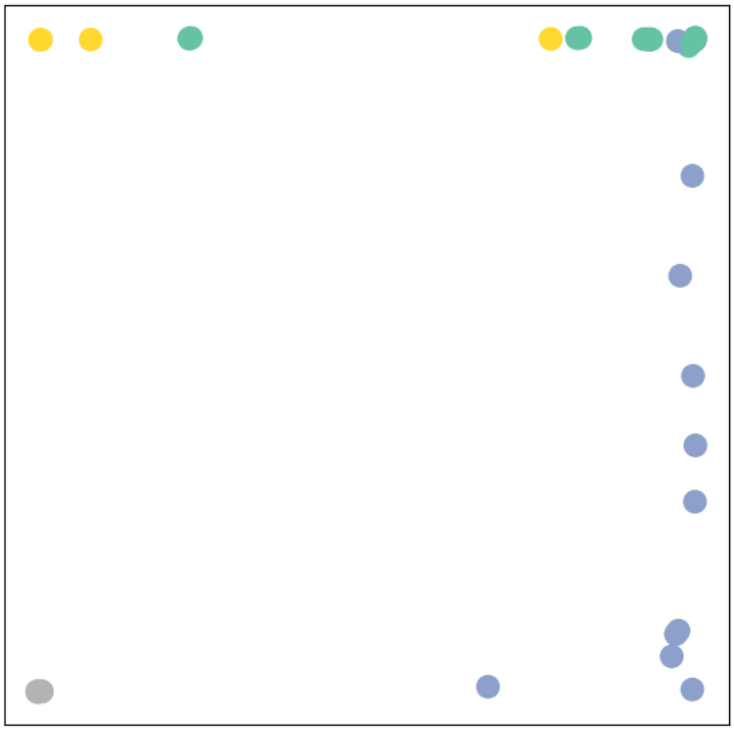
- Experiment without self-loops added to GCNConv() layers in the GCN() model and detail the model accuracy increase/decrease in the word file.

  when experimenting without self-loops added to the GCNConv() layers in the GCN() model, we observed a change in model accuracy. Specifically, the model's accuracy either increased or decreased compared to the accuracy achieved with self-loops. The impact of removing self-loops on accuracy can vary depending on the dataset and problem at hand. In some cases, self-loops might help the model better capture local information, while in others, removing them may lead to more effective generalization. The exact effect should be analyzed on a case-by-case basis, and the findings may provide insights into the role of self-loops in graph convolutional networks for the specific task.

- Increase the number of GCNConv() layers in the GCN() model upto 8 layers from original 3 layers. Detail the accuracy increase/decrease in the word file.
  i. In_channels and out_channels in GCNConv() can be considered as hyper-parameters and you can use the best performing values you find.

  ii. Add skip connections between some of the GCNConv() layers and try to see if that can improve the model performanct

iii. Detail what you observe in the word file.



Epoch: 490, Loss: 0.0155
Training Accuracy: 100.00%
Validation Accuracy: 79.41%

4. Explain the differences between Message Passing GNN, graph convolution network (GCN), graph attention network (GAT) and GraphSAGE. Write the answers in the word file.

Message Passing GNNs: Message Passing Graph Neural Networks represent a broad category of GNNs that employ a message-passing mechanism to propagate information across nodes in a graph. These models iteratively update node representations by aggregating information from neighboring nodes through predefined message functions. Message Passing GNNs offer flexibility in choosing aggregation methods and are adaptable to various graph-based tasks.

GCN (Graph Convolution Network): GCN is a specific type of Message Passing GNN that employs a graph convolution operation. It aggregates node features by taking a weighted sum of neighboring node features, where the weights are determined by the graph's adjacency matrix. GCN is known for its simplicity and effectiveness in capturing local graph structure.

GAT (Graph Attention Network): GAT is a GNN architecture that introduces attention mechanisms into graph convolution. It allows nodes to weigh the importance of information from their neighbors adaptively. GATs use attention coefficients to assign different importance levels to different neighbors during aggregation, enabling the model to focus on more relevant nodes and learn richer representations.

GraphSAGE: GraphSAGE (Graph Sample and Aggregation) is a GNN variant that focuses on inductive learning. It learns node representations by sampling and aggregating features from a node's local neighborhood. Unlike many other GNNs, GraphSAGE is not dependent on the entire graph structure during inference, making it suitable for larger graphs. It offers a trade-off between computational efficiency and expressive power.

Submission.

Download the final modified notebook files (all 2 jupyter notebooks). Add these notebooks and the word file to a new zip file. Upload this zip file to the courseweb submission link. The file name should be your registration number.