GitHub repo: <https://github.com/IT21226632/DL_Lab_06.git>

1. 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).

When increasing the number of nodes from 20 to 200 in a random graph, the graph density remains relatively consistent because the probability of edge formation stays the same, regardless of the number of nodes. However, the degree distribution changes significantly.

For smaller graphs (e.g., with 20 nodes), the degrees (number of connections per node) tend to be concentrated around a narrow range. There are fewer nodes to connect to, and randomness plays a smaller role in shaping the overall structure.

The degree distribution broadens as the number of nodes increases (e.g., with 200 nodes). More nodes can connect to more neighbors, leading to a wider variety of node degrees. You'll see more nodes with both very few and many connections, though the average number of connections tends to stabilize around a specific value. The overall structure becomes more complex, with a larger pool of possible connections creating a more diverse network.

**Supervised, Self-Supervised, and Semi-Supervised Learning**

1. **Supervised Learning**
   * Uses labeled data to train the model.
   * Example: Image classification where each image has a label (e.g., cat or dog).
2. **Self-Supervised Learning**
   * Model creates its own labels from the data to learn useful representations.
   * Example: Predicting the next word in a sentence.
3. **Semi-Supervised Learning**
   * Combines a small amount of labeled data with a large amount of unlabeled data.
   * Example: Classifying images when only a few are labeled.

**Inductive vs. Transudative Learning**

1. **Inductive Learning**
   * Learns a general function that can be applied to unseen data.
   * Example: Training on a dataset and applying the model to new, unseen instances.
2. **Transductive Learning**
   * Makes predictions only for known test data, without generalizing to new data.
   * Example: Predicting labels for specific nodes in a graph using both labeled and unlabeled nodes.

3.1 Increase the number of epochs from 50 to 500, observe the change in validation accuracy, and write what you observe in the word file.

A graph showing a number of data

Description automatically generated with medium confidenceA graph showing a number of dots

Description automatically generated with medium confidence

Figure 2Accuracy when the epocs 500

Figure 1Accuracy when the ecpocs 50

The validation accuracy steadily improves from 64% at 50 epochs to 82% at 500 epochs, indicating consistent learning over time without significant signs of overfitting.

The model can learn and generalize to unseen data, as the validation accuracy increases with more epochs.

While the rate of accuracy improvement slows down after epoch 300, the final accuracy of 82% at 500 epochs shows that the model continues to improve without memorizing the training data.

The model demonstrates effective training with a well-balanced learning rate, as it captures more patterns from the data without fitting specific noise or overfitting.

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

Figure 4when epocs 500

Figure 3when ecpos 50

A graph showing a graph of a graph

Description automatically generated with medium confidenceA graph showing a number of points

Description automatically generated with medium confidence

Removing self-loops leads to a drop in validation accuracy across all epochs, with the largest impact seen in earlier epochs.

At 50 epochs, the accuracy drops to 47% without self-loops, compared to 64% with self-loops.

By 500 epochs, the accuracy improves to **61%**, but it still lags significantly behind the 82% observed with self-loops, indicating that the model is less effective without them.

Self-loops are important because they allow nodes to preserve their own features while aggregating information from neighboring nodes. Without self-loops, the GCN struggles to learn representations, leading to reduced accuracy effectively.

The overall performance is weaker without self-loops, suggesting that they are essential for improving the model’s ability to generalize.

A white background with black text

Description automatically generated3.3. 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.

Figure 6epocs 500

A screen shot of a graph

Description automatically generated

Figure 5when epocs 50

Increasing only the layers without changing the channel counts allows the model to propagate information across a wider neighborhood in the graph but can lead to issues like over-smoothing, limited capacity for complex features, and possible performance degradation.

The trade-off is that adding more layers increases model depth, but without increasing feature richness (channel count), the model might not fully benefit from the added depth.

3.3.1 In\_channels and out\_channels in GCNConv() can be considered as hyper-parameters and you can use the best-performing values you find.

The first two layers (conv1 and conv2) have larger channel sizes (32), enabling the network to learn a rich set of features from the input.

As the model goes deeper, the number of channels decreases (from 32 → 16 → 8 → 4 → 2). This gradual reduction helps compress the features and prevents the model from overfitting, as deeper layers focus on learning more abstract representations.

The chosen configuration with 32 → 32 → 16 → 16 → 8 → 8 → 4 → 2 channels strikes a good balance between model depth and feature compression, providing the best performance for this task.

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

Skip connections prevent the model from losing important information from the earlier layers and mitigate the vanishing gradient problem. This approach can improve performance and stability, especially in deep GNNs.