Lab 6

1.

The graph density gradually drops as the number of nodes (N) in the random graph rises from 20 to 200. This happens because as N increases, the graph becomes sparser because the number of potential edges increases more quickly than the number of actual edges. The degree distribution is more uniform and the density is higher in smaller graphs (such N = 20 N = 20) when the majority of nodes have around the same amount of connections. But as N N rises, the degree distribution gets more dispersed, which causes the number of connections per node to vary considerably, with some nodes developing into highly linked hubs and others staying sparsely connected. As a result, the overall graph structure becomes sparser and the degree distribution becomes larger.

2.

Differences between Supervised Learning, Self-Supervised Learning, and Semi-Supervised Learning:

- Supervised Learning:

  In supervised learning, a model is trained on a labeled dataset where both the input data and the corresponding labels are provided. The goal is to learn a mapping from inputs to outputs so the model can predict labels for unseen data. This method requires large amounts of labeled data, which can be expensive and time-consuming to obtain.

  - **Example:** Classifying images of cats and dogs, where each image has a corresponding label (cat or dog).

- **Self-Supervised Learning:**

  Self-supervised learning leverages unlabeled data by generating labels from the data itself. The model learns representations from these pseudo-labels and uses them to improve performance on downstream tasks. It reduces the need for manually labeled data by using inherent patterns in the data for supervision.

  - **Example:** Predicting the next word in a sentence or filling in missing parts of an image.

- **Semi-Supervised Learning:**

  Semi-supervised learning is a hybrid approach that combines a small amount of labeled data with a large amount of unlabeled data. The model first learns from the labeled data and then uses the unlabeled data to further refine its predictions. This approach reduces the reliance on large labeled datasets while still benefiting from the presence of some labeled examples.

  - **Example:** Training a model on a dataset where only a portion of the images are labeled, and the rest are unlabeled.

Differences between Transductive Learning and Inductive Learning:

- **Transductive Learning:**

  In transductive learning, the model is trained on a specific training dataset, and predictions are only made for the particular test data provided during training. The model doesn't generalize to unseen data outside this specific test set. This approach is used in scenarios where all test data is available at the time of training.

  - Example: Graph Neural Networks (GCNs) on node classification, where both labeled and unlabeled nodes in the graph are available during training, and predictions are made for the unlabeled nodes.

- **Inductive Learning:**

  In inductive learning, the model learns a general function from the training data that can be applied to make predictions on unseen, new data that wasn't available during training. The goal is to generalize beyond the specific dataset used in training to other unseen datasets.

  - Example: Training a machine learning model to classify images of animals and then using it to classify images of animals it has never seen before.

3.

Increasing the number of epochs from 50 to 500:

When the number of epochs is increased from 50 to 500, the validation accuracy generally improves initially. As the model trains for more epochs, it learns better representations from the graph data. However, after a certain point, overfitting may occur, and the improvement in validation accuracy slows down or becomes negligible. The early epochs are more critical for rapid gains in accuracy, and the later epochs show diminishing returns.

Experiment without self-loops in GCNConv() layers:

Self-loops in GCNConv() layers allow each node to consider its own features during the convolution operation. Removing self-loops leads to a slight decrease in model accuracy because the model loses the ability to capture important information from each node's own features. As a result, nodes rely only on neighboring nodes' features, which can lead to reduced learning of local node characteristics and a slight performance drop.

Increasing the number of GCNConv() layers to 8 from the original 3 layers:

Increasing the number of layers in the GCN model can initially improve accuracy because the model can learn more complex graph structures. However, beyond a certain number of layers (in this case, increasing up to 8 layers), the model might experience over-smoothing, where node features become too similar across the graph, causing the model's performance to degrade. This leads to a decrease in validation accuracy as deeper layers introduce diminishing returns or even hinder learning.

Tuning the in_channels and out_channels hyperparameters:

Adjusting the in_channels and out_channels hyperparameters in GCNConv() affects the model's performance. By increasing the number of channels, the model's capacity to capture graph information improves, resulting in higher accuracy. However, excessively large channel numbers can introduce overfitting or increase computational costs, so finding an optimal balance is crucial.

Adding skip connections between GCNConv() layers:

Adding skip (residual) connections between some of the GCNConv() layers improves the model's performance by alleviating the vanishing gradient problem and allowing the model to retain information from earlier layers. This prevents over-smoothing in deeper layers and results in improved accuracy, as the model benefits from both shallow and deep layer features. Skip connections help in stabilizing training and improving generalization, especially with deeper GCN architectures.

4.

- **Message Passing GNN (MP-GNN):**

  Message Passing GNN is a broad framework used for graph learning tasks where nodes in a graph communicate with their neighbors iteratively. In each iteration (or layer), a node aggregates information from its neighbors by passing and receiving messages. This aggregated information is then used to update the node's features. The process is repeated for a predefined number of iterations, allowing nodes to gather information from increasingly distant neighbors. MP-GNN is flexible and can be customized for a variety of applications like node classification, link prediction, and graph classification. The operations for message passing and aggregation are key to defining how information flows in the graph.

- **Graph Convolutional Network (GCN**):

  GCN is a specific kind of MP-GNN that generalizes convolution operations from grid data (as in images) to graph-structured data. In GCN, each node aggregates information from its neighbors by taking a weighted sum of their features, akin to how convolution in image processing aggregates pixel information. A shared weight matrix is used across all nodes, which reduces complexity and prevents overfitting. GCNs are particularly effective for semi-supervised learning tasks on graphs, such as node classification. However, GCNs can face challenges when deeper layers are added, leading to over-smoothing, where node features become indistinguishable as more layers are applied.

- **Graph Attention Network (GAT):**

  GAT introduces an attention mechanism to the GCN framework to assign different weights to neighboring nodes during aggregation. In traditional GCN, all neighbors are treated equally, but GAT allows the model to learn which neighbors are more important by learning attention coefficients for each edge during training. This makes GAT more flexible and powerful in scenarios where the significance of neighbors varies, especially in heterogeneous graphs. By focusing more on relevant neighbors, GAT improves the accuracy of tasks like node classification while maintaining the advantages of convolutional layers. GAT can also handle self-loops to allow nodes to retain their own features in the aggregation process.

- **GraphSAGE:**

  GraphSAGE addresses the scalability limitations of GCN by using a neighborhood sampling strategy. Instead of aggregating features from all of a node's neighbors (which may be computationally expensive in large graphs), GraphSAGE samples a fixed number of neighbors, allowing it to scale more efficiently to larger datasets. This sampling is combined with aggregation functions like mean, LSTM-based aggregators, or pooling techniques to update node features. Additionally, GraphSAGE is designed for inductive learning, meaning it can generalize to unseen nodes or even entirely new graphs, unlike GCN, which is typically used in a transductive setting. This makes GraphSAGE particularly suitable for large, evolving graphs.