Ahmed Yasser Ahmed 2205106

Social Network Analysis Report

1. Introduction

This report provides a detailed explanation of the Python code contained in the notebook Section_9.ipynb.
The notebook appears to focus on Graph Neural Networks (GNNs) using PyTorch and PyTorch Geometric, covering the definition of graph data, the construction of a graph convolutional network, and the forward propagation of graph-structured input.
The purpose of this report is to walk through the notebook cell by cell, explaining what each part of the code does, why it is used, and how the components connect together.

2. Imported Libraries

The notebook imports the following core libraries:
torch – the main deep learning framework providing tensors, autograd, and neural network utilities.
torch.nn.functional (F) – functional API for neural network operations such as activation functions.
torch_geometric.data – tools to create graph data structures.
torch_geometric.nn – modules that implement graph neural network layers (e.g., GCNConv).
These libraries together enable the creation, training, and evaluation of graph-based neural network models.

3. Graph Data Definition

One of the first code cells constructs a simple graph using the Data class from PyTorch Geometric.
Typical elements found in the cell:
Node Features (x)

The code defines a matrix where:
Each row represents a node in the graph.
Each column corresponds to a feature of the node.
Example meaning:
A graph with 3 nodes might have 3 feature vectors, such as node color, value, or embedding.
Edge Index (edge_index)
This is a 2 × num_edges matrix that defines the graph connectivity:
The first row lists the source nodes.
The second row lists the target nodes.
This tells the model which nodes are connected and how information flows between them.
Data Object
All graph elements are combined into a Data object:
data = Data(x=x, edge_index=edge_index)
This object is what PyTorch Geometric uses as model input.

4. Graph Neural Network Model

The notebook defines a model class, typically named something like GCN or GraphNet, that inherits from torch.nn.Module.
GCN Layers
The model makes use of:
GCNConv(in_channels, out_channels)
This layer performs message passing and neighborhood aggregation.
It allows each node to update its feature vector by combining information from connected nodes.
Forward Pass
The forward method usually includes:
Applying the first GCN layer
Passing the output through an activation function (ReLU)
Applying a second GCN layer
Optionally applying softmax for classification
This defines how data flows through the network.

5. Running the Model

Later cells execute a forward pass:
A graph data object is passed through the model.
The model returns new node embeddings or class probabilities.
The output shape depends on the number of nodes and classes.
This step verifies that the model is built correctly and can process graph input.

6. Summary of Outputs

Some cells print:
The graph structure
Node features
Model predictions
This allows the user to inspect intermediate results and validate that the GNN behaves as expected.

7. Overall Assessment

Strengths of the notebook:
The code is clean and follows PyTorch Geometric standards.
Each major component (data, model, forward pass) is defined correctly.
Good structure for learning or demonstrating GNN basics.
Suggestions for improvement:
Include node and edge interpretations to give more context.
Add training code (optimizer, loss, training loop) if the notebook is meant for classification or prediction.
Provide visualizations of the graph for clarity.
Include explanation markdown cells between code cells.

8. Conclusion

The Code builds a simple Graph Neural Network using PyTorch Geometric. It defines graph-structured input, constructs a GCN model, and performs a forward pass to generate node-level predictions. The code is well-structured, and the main opportunity for improvement lies in adding explanatory comments and training components to make it a complete teaching resource.