Mathematical Formulation of Graph Neural Networks

Sidharth SS (IISER, Mohali)

1 Introduction

Graph Neural Networks (GNNs) have emerged as a powerful class of models for learning on graph-structured data. A graph is defined as

$$G = (V, E),$$

where V is a set of nodes (or vertices) and E is a set of edges that encode relationships between nodes. Many real-world problems—from social network analysis to molecular property prediction—naturally involve graphs. This document provides a detailed mathematical framework for understanding and implementing GNNs, emphasizing the message passing paradigm.

2 Graph Representations

A graph G is often represented by several components:

- **Nodes:** Each node $i \in V$ is associated with a feature vector $\boldsymbol{x}_i \in \mathbb{R}^d$, where d is the number of features.
- **Edges:** An edge $e_{ij} \in E$ connects nodes i and j. Edges can also carry features (e.g., bond types in molecules).
- **Adjacency Matrix:** The connectivity of the graph is captured by the matrix

$$A \in \mathbb{R}^{N \times N}, \quad A_{ij} = \begin{cases} 1, & \text{if there is an edge between } i \text{ and } j, \\ 0, & \text{otherwise.} \end{cases}$$

  To ensure that a node's own features are included in the update, self-loops are typically added:

$$\tilde{A} = A + I_N,$$

  where I_N is the $N \times N$ identity matrix.

3 The Message Passing Paradigm

The core of GNNs is the message passing mechanism, which updates node representations by exchanging information with their neighbors. For each node i, the update is broken into three main steps:

3.1 Message Function

Each node $j \in \mathcal{N}(i)$ (where $\mathcal{N}(i)$ denotes the neighborhood of node i) sends a message to node i. The message from node j is given by:

$$m_{ij} = F(\boldsymbol{x}_j),$$

where F is a learnable function. A common choice for F is an affine transformation:

$$m_{ij} = W\boldsymbol{x}_j + b,$$

with weight matrix $W \in \mathbb{R}^{d' \times d}$ and bias $b \in \mathbb{R}^{d'}$. For notational simplicity, the bias is sometimes omitted.

3.2 Aggregation Function

Once the messages $\{m_{ij} : j \in \mathcal{N}(i)\}$ are computed, they are aggregated using a permutation-invariant function G:

$$a_i = G\Big(\{m_{ij} : j \in \mathcal{N}(i)\}\Big).$$

A common aggregation is the summation:

$$a_i = \sum_{j \in \mathcal{N}(i)} m_{ij}.$$

Other choices include averaging or taking the maximum.

3.3 Update Function

Finally, the node's own feature vector is updated by combining its current state with the aggregated message:

$$\boldsymbol{x}_i' = U(\boldsymbol{x}_i, a_i).$$

A typical formulation applies an activation function $\sigma(\cdot)$ after combining the contributions:

$$\boldsymbol{x}_i' = \sigma\Big(H(\boldsymbol{x}_i) + K(a_i)\Big),$$

or, in a more streamlined version:

$$\boldsymbol{h}_i' = \sigma\Big(W_{\text{self}}\boldsymbol{h}_i + \sum_{j \in \mathcal{N}(i)} W_{\text{neigh}}\boldsymbol{h}_j\Big),$$

where W_{self} and W_{neigh} are learnable parameters.

4 Vectorized Formulation

For efficient implementation, the node features are stacked into a matrix:

$$X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} \in \mathbb{R}^{N \times d}.$$

The affine transformation for all nodes can then be written as:

$$XW \quad \text{with} \quad W \in \mathbb{R}^{d \times d'}.$$

Incorporating the augmented adjacency matrix $\tilde{A}$, the message passing step for the entire graph becomes:

$$X' = \sigma\Big(\tilde{A}XW\Big).$$

This formulation ensures that each node aggregates information from its neighbors as well as itself.

A popular variant, introduced by Kipf and Welling in their Graph Convolutional Network (GCN), includes a normalization step. Define the degree matrix $\tilde{D}$ for $\tilde{A}$ as:

$$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}.$$

Then, the layer-wise propagation rule becomes:

$$X' = \sigma\Big(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}XW\Big).$$

5 GNN Applications

The mathematical formulation of GNNs underpins several applications:

5.1 Graph-Level Representation

To obtain a compact representation of the entire graph (useful for graph classification), a pooling function is applied over all node embeddings:

$$h_G = \text{Pool}(X') = \text{Aggregate}\{\boldsymbol{x}'_1, \boldsymbol{x}'_2, \ldots, \boldsymbol{x}'_N\}.$$

5.2 Node Classification

For tasks such as node classification, each updated node embedding $\boldsymbol{x}'_i$ is passed through a classifier:

$$\hat{y}_i = \text{softmax}\Big(f(\boldsymbol{x}'_i)\Big),$$

where f is a function (typically a linear layer) mapping the node embedding to a K-dimensional output corresponding to the K classes.

5.3 Link Prediction

In link prediction tasks, the goal is to determine whether an edge should exist between nodes i and j. This can be achieved by combining their embeddings:

$$p_{ij} = \sigma\Big(g(\boldsymbol{x}'_i \,\|\, \boldsymbol{x}'_j)\Big),$$

where $\|$ denotes concatenation, g is a learnable function (such as a multi-layer perceptron), and σ is typically a sigmoid function producing a probability.

6 Conclusion

This document has presented a detailed and rigorous mathematical exposition of Graph Neural Networks. We started from the basic graph representation and built up through the message passing paradigm, culminating in a vectorized formulation suitable for efficient implementation. These principles form the backbone of many modern applications that require learning on non-Euclidean domains.