Name:Rewan ahmed Elwardany
ID:2205218

Report:the GraphSAGE Node Classification Code

---

1. Introduction

The following code represents the implementation of a GNN using GraphSAGE from the PyTorch Geometric library.

The goal of this model is to classify each node in a small graph as either benign (0) or malicious (1) based on both their features and their structural connections.

The following report explains each part of the code and what concepts it demonstrat

---

2. Graph Construction

2.1 Node Features

The graph has 6 nodes, and each of them features 2 features:

Benign nodes → feature vector: [1.0, 0.0]

The malicious nodes → feature vector: [0.0, 1.0]

It can, that is, tell the types of the nodes given the input features.

2.2 Edges

The edges connect the nodes as follows:

These are benign nodes that are connected to each other, or, in other words, are densely connected together: 0,1,2.

Malicious nodes (3,4,5) are also densely connected.

A single cross-community edge is present between node 2 → node 3.

All the edges are added twice, that is 0→1 and 1→0, to make an undirected graph.

2.3 Labels

Nodes are labeled depending on their type:

Nodes 0, 1, 2 → label 0 (benign)

Nodes 3, 4, 5 → label 1 - malicious

These become the labels that will train the classifier.

---

3. Creating the Data Object

The code creates a torch_geometric.data.Data object that contains:

x → node features

edge_index → graphics edges

y → true labels for training

This is the standard input format for Graph Neural Networks in PyTorch Geometric.

---

4. Defining the GraphSAGE Model

We will be using a two-layer GraphSAGE neural network model that will be implemented as a PyTorch class.

4.1 Layer Structure

Layer 1: SAGEConv(2 → 4)

Aggregates neighbor information and constructs 4-dimensional hidden embeddings.

ReLU activation: introduces nonlinearities.

Layer 2: SAGEConv(4 → 2)

Produces final class scores for two classes: benign and malicious.

Log Softmax: turns raw outputs into log-probabilities.

GraphSAGE is designed to work by sampling and averaging information from neighbors, so node classification depends both on graph structure and features.

---

5. Training the Model

A simple training loop could run for 50 epochs:

Forward pass: compute predictions

Now calculate the NLLLoss() w.r.t actual labels

Backpropagate gradients

Weight update using Adam optimizer Because this is a small demonstration graph, all nodes are used both for training and evaluation

---

6. Making Predictions

pred = model(data.x, data.edge_index).argmax(dim=1) This outputs the predicted class-0 or 1-for each node. The model will be able to learn the correct classification between benign versus malicious nodes by using their features and neighbor patterns.