Name:Farah Mahmoud Elhenawy

ID:2205199

1. Importing Required Libraries

```
!pip install torch_geometric

import torch
from torch_geometric.nn import SAGEConv
import torch.nn.functional as F
```

- **PyTorch** is used for tensor operations and model training.
- **PyTorch Geometric (PyG)** provides graph neural network (GNN) layers.
- **SAGEConv** is the GraphSAGE convolution layer.
- **F** contains activation functions and softmax utilities.

---

2. Defining the Graph Structure

```
x = torch.tensor(
    [
        [1.0, 0.0],  # Node 0 (benign)
        [0.8, 0.1],  # Node 1 (benign)
        [0.9, 0.2],  # Node 2 (benign)
        [0.1, 0.9],  # Node 3 (malicious)
        [0.2, 0.8],  # Node 4 (malicious)
        [0.3, 1.0]   # Node 5 (malicious)
    ],
    dtype=torch.float,
)
```

- Creates **6 nodes**, each with **2 features**.
- First three nodes represent **benign users**, last three represent **malicious users**.
- These features could represent any meaningful signals (behavior metrics, profile embeddings).

In a real misinformation detection dataset, these features would come from user activity or content embedding.

---

3. Defining Edges (Connections Between Nodes)

```
edge_index = (
    torch.tensor(
        [
            [0, 1, 1, 2, 3, 4, 4, 5],  # source nodes
            [1, 0, 2, 1, 4, 3, 5, 4]   # target nodes
        ],
        dtype=torch.long,
    )
    .t()
    .contiguous()
)
```

- `edge_index` describes **which nodes are connected**.
- The graph is **undirected** (connections appear in both directions).
- Nodes 0–2 are densely connected → **benign cluster**
- Nodes 3–5 are connected → **malicious cluster**

This is a simplified representation of two communities.

---

4. Node Labels

```
y = torch.tensor([0, 0, 0, 1, 1, 1], dtype=torch.long)
```

- Each node is assigned a **class label**:
  - `0 = benign`
  - `1 = malicious`
- This is a supervised learning setup for **node classification**.

---

5. Defining the GNN Model (GraphSAGE)

```
class GraphSAGE(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = SAGEConv(2, 4)
        self.conv2 = SAGEConv(4, 2)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

- The model has **two GraphSAGE layers**:
  1. `2 → 4` transforms node features into a 4-dimensional hidden space.

    2. `4 → 2` outputs class logits (benign vs malicious).
- **ReLU** adds nonlinearity.
- **log_softmax** prepares output for classification loss.

GraphSAGE aggregates information from neighboring nodes to learn community patterns.

---

6. Training the Model

```
model = GraphSAGE()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

for epoch in range(100):
    optimizer.zero_grad()
    out = model(x, edge_index)
    loss = F.nll_loss(out, y)
    loss.backward()
    optimizer.step()
```

- An **Adam optimizer** is used.
- Training loop runs **100 epochs**.
- Steps in each epoch:
    1. **Forward pass** → compute predictions.
    2. **Loss calculation** (`nll_loss` for log probabilities).
    3. **Backward pass** → compute gradients.
    4. **Optimization update**.

The model learns to classify nodes based on both their features and graph connections.

---

7. Model Output and Interpretation

At the end of training, the model produces a **2-class probability** for each node.
The higher probability corresponds to the predicted class (benign or malicious).

This demonstrates how GNNs can detect **misinformation communities, bot clusters, or user groups** using network structure.

The code implements a Graph Neural Network (GNN) using the GraphSAGE architecture to classify nodes in a small social-network graph as either *benign* or *malicious*. Each node is represented with a 2-dimensional feature vector, and the edges represent relationships between users. The model uses two GraphSAGE convolution layers to aggregate information from neighboring nodes, allowing the classifier to learn community patterns. The training is performed using supervised learning with cross-entropy loss. This approach demonstrates how GNNs can detect clusters or communities associated with misinformation, bots, or coordinated malicious behavior by leveraging both node attributes and network connectivity.
