GraphSAGE Node Classification

Name : Kerelos Zakarya Tadros

Student ID : 2205191

Installing Required Libraries

The project begins by installing and importing the necessary libraries used to build a GNN model.
Main libraries used include:

- **torch**
- **torch_geometric**
- **SAGEConv (GraphSAGE convolution layer)**

```python
import torch
from torch_geometric.data import Data
from torch_geometric.nn import SAGEConv
import torch.nn.functional as F
```

2. Defining Node Features

Six nodes are created, each represented by a 2-dimensional feature vector:

- **[1, 0] → benign node**
- **[0, 1] → malicious node**

This forms the input feature matrix:

```python
x = torch.tensor([
    [1.0, 0.0],
    [1.0, 0.0],
    [1.0, 0.0],
    [0.0, 1.0],
    [0.0, 1.0],
    [0.0, 1.0],
], dtype=torch.float)
```

3. Creating Graph Edges

Two clusters are modeled:

Benign subgraph (nodes 0–1–2)

Fully connected.

Malicious subgraph (nodes 3–4–5)

Fully connected.

One cross-edge between groups

- **Node 2 → 3 links benign to malicious.**

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

5. Building the GraphSAGE Model

A two-layer GraphSAGE neural network is defined with:

- **2 input features**
- **4 hidden units**
- **2 output classes (benign vs malicious)**

Model code:

```
# GraphSAGE model
class GraphSAGENet(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super().__init__()
        self.conv1 = SAGEConv(in_channels, hidden_channels)
        self.conv2 = SAGEConv(hidden_channels, out_channels)

    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)

```

Training the Model and Prediction

The model is trained using:

- **Adam optimizer**
- **Learning rate = 0.01**
- **50 epochs**
- **NLLLoss as the loss function**

```
41  # Training
42  optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
43  model.train()
44  for epoch in range(50):
45      optimizer.zero_grad()
46      out = model(x, edge_index)
47      loss = F.nll_loss(out, y)
48      loss.backward()
49      optimizer.step()
50
51  # Prediction
52  model.eval()
53  pred = model(x, edge_index).argmax(dim=1)
54  print("Predicted labels:", pred.tolist())
```

Output:

Predicted labels: [0, 0, 0, 1, 1, 1]