social network report of code

eman ahmed 2205143

1. Introduction

The provided code demonstrates how to build and train a simple **Graph Neural Network (GNN)** using **GraphSAGE** from the PyTorch Geometric library.

The goal of the model is to classify nodes in a small graph as either:

- **Benign (label = 0)**
- **Malicious (label = 1)**

The graph contains **6 nodes**, each with **two features**, and is connected by an adjacency list (edge list).

After training, the model predicts the class of each node.

2. Installing and Importing Required Libraries

```
!pip install torch_geometric
import torch
from torch_geometric.data import Data
from torch_geometric.nn import SAGEConv
import torch.nn.functional as F
```

Explanation

- `pip install torch_geometric` → installs PyTorch Geometric library.
- `import torch` → loads PyTorch for tensors and neural networks.
- `from torch_geometric.data import Data` → Data object that stores the graph (features, edges, labels).
- `from torch_geometric.nn import SAGEConv` → GraphSAGE convolution layer.
- `import torch.nn.functional as F` → provides activation functions and loss functions.

3. Defining Node Features

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

Explanation

- Each node has **2 feature values**.
- Benign nodes are represented as: **[1, 0]**
- Malicious nodes are represented as: **[0, 1]**
- These features help the model learn patterns during training.

4. Defining Graph Edges (Connections)

```
edge_index = (
    torch.tensor(
        [
            [0, 1], [1, 0],
            [1, 2], [2, 1],
            [0, 2], [2, 0],   # benign cluster 0-1-2 fully connected
            [3, 4], [4, 3],
            [4, 5], [5, 4],
            [3, 5], [5, 3],   # malicious cluster 3-4-5 fully connected
            [2, 3], [3, 2],   # one edge connecting benign to malicious
        ],
```

```
        dtype=torch.long,
    ).t()
)
```

Explanation

- `edge_index` is a **2 × E** matrix containing graph edges.
- `.t()` transposes it so format becomes:

```
[[source nodes],
 [target nodes]]
```

- Edges are **undirected**, so each edge appears twice (e.g., 0→1 and 1→0).
- The graph has:
  - A **benign cluster** (nodes 0,1,2 fully connected)
  - A **malicious cluster** (nodes 3,4,5 fully connected)
  - One connecting edge: **2 ↔ 3**

This structure helps the GNN learn community patterns.

5. Defining Node Labels

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

Explanation

Mapping:

- Nodes 0,1,2 → benign → label 0
- Nodes 3,4,5 → malicious → label 1

This is the target data the model tries to learn.

6. Creating the Graph Data Object

```
data = Data(x=x, edge_index=edge_index, y=y)
```

Explanation

The `Data` object holds everything PyTorch Geometric needs:

- `x`: node features
- `edge_index`: graph edges
- `y`: node labels

7. Defining the GraphSAGE Model

```
class GraphSAGENet(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GraphSAGENet, self).__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)   # 1st GraphSAGE layer
        x = F.relu(x)                   # activation
        x = self.conv2(x, edge_index)   # 2nd GraphSAGE layer
        return F.log_softmax(x, dim=1)  # output class probabilities
```

Explanation of Each Part

- `in_channels=2` → because each node has 2 features.
- `hidden_channels=4` → first layer outputs 4-dimensional embeddings.
- `out_channels=2` → two classes: benign & malicious.

Forward Pass

1. The first GraphSAGE layer aggregates neighbor features.
2. ReLU adds non-linearity.

3. Second SAGE layer produces final output.
4. `log_softmax` gives class probabilities for each node.

8. Training the Model

```
model = GraphSAGENet(in_channels=2, hidden_channels=4, out_channels=2)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
model.train()

for epoch in range(50):
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)
    loss = F.nll_loss(out, data.y)
    loss.backward()
    optimizer.step()
```

Explanation

- The model is trained for **50 epochs**.
- Steps each epoch:
    1. **zero_grad()** → clears old gradients
    2. **model()** → computes predictions
    3. **loss = nll_loss(...)** → calculates classification loss
    4. **backward()** → computes gradients
    5. **optimizer.step()** → updates weights

The model gradually learns node classification.

9. Making Predictions

```
model.eval()
pred = model(data.x, data.edge_index).argmax(dim=1)
```

```
print("Predicted labels:", pred.tolist())
```

Explanation

- `model.eval()` → evaluation mode (no gradients).
- `argmax(dim=1)` → picks the most likely class (0 or 1).

Output

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

The model correctly learned to classify benign and malicious nodes.