GraphSAGE Code Walkthrough

Student Name: Mohamed Abdelrahman Awad Kaled **Student ID:** 2205114

This is a breakdown of the GraphSAGE code, explaining what each part does and why it's there.

1. Setup and Imports

We need PyTorch and the PyTorch Geometric (PyG) library for graph-based deep learning.

```
!pip install torch_geometric

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

- **`!pip install torch_geometric`** : Installs the PyG library.
- **`import torch`** : Imports the core PyTorch library.
- **`from torch_geometric.data import Data`** : Imports the `Data` object, which organizes the graph (features, edges, labels).
- **`from torch_geometric.nn import SAGEConv`** : Imports the GraphSAGE convolution layer, which handles neighbor aggregation.
- **`import torch.nn.functional as F`** : Imports functions for activation (ReLU) and loss calculation.

2. Defining the Data: User Features (`x`)

We define 6 users (nodes) and their initial features.

```
#--- Define a small graph with 6 nodes ---
# Node features (2 features per node).
# Here benign users have [1, 0] and malicious have [0, 1] for illustration.
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,
)
```

- **The `x` tensor**: The feature matrix (6 rows for users, 2 columns for features).
- **Feature Logic**:
  - Users 0, 1, 2 (Good) get `[1.0, 0.0]`.
  - Users 3, 4, 5 (Bad/Fraud) get `[0.0, 1.0]`.
  - This is a simple binary feature setup for demonstration.

---

3. Defining the Data: Connections (`edge_index`)

This defines the graph structure—who is connected to whom.

```
# Edge list (undirected). Connect benign users (0-1-2 fully connected)
# and malicious users (3-4-5 fully connected), plus one cross-edge 2-3.
edge_index = (
    torch.tensor(
        [
            [0, 1],
            [1, 0],
            [1, 2],
            [2, 1],
            [0, 2],
            [2, 0],
            [3, 4],
            [4, 3],
            [4, 5],
            [5, 4],
            [3, 5],
            [5, 3],
            [2, 3],
            [3, 2],  # one connection between a benign (2) and malicious (3)
        ],
        dtype=torch.long,
    )
    .t()
    .contiguous()
)
```

- **The `edge_index` tensor**: A list of connections.
- **Structure**:
  - Good users (0, 1, 2) are fully connected.
  - Bad users (3, 4, 5) are fully connected.
  - **Crucial Link**: A single connection exists between user 2 (Good) and user 3 (Bad) (`[2, 3]` and `[3, 2]`). This tests the model's ability to handle mixed neighborhoods.
- **`.t().contiguous()`** : Formats the edge list for PyG.

---

4. The GraphSAGE Model and Data Object

We define the ground truth labels and the neural network architecture.

```
# Labels: 0 = benign, 1 = malicious
y = torch.tensor([0, 0, 0, 1, 1, 1], dtype=torch.long)

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

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

- **Labels (`y`)**: The ground truth: `[0, 0, 0]` (Good) and `[1, 1, 1]` (Bad).
- **`data = Data(...)`** : Bundles features, connections, and labels.
- **The `GraphSAGENet` Class**: A two-layer GraphSAGE model.
    - **`__init__`** : Sets up two `SAGEConv` layers with dimensions `in=2` , `hidden=4` , and `out=2` (for two classes).
    - **`forward`** : Defines the data flow: Convolution -> ReLU activation -> Second Convolution -> `F.log_softmax` for final log-probabilities.

5. Training and Checking the Results

The standard deep learning process to train the model and generate predictions.

```
# Instantiate model: input dim=2, hidden=4, output dim=2
model = GraphSAGENet(in_channels=2, hidden_channels=4, out_channels=2)

# Simple training loop
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()
```

```
# After training, we can check predictions
model.eval()
pred = model(data.x, data.edge_index).argmax(dim=1)
print("Predicted labels:", pred.tolist())
```

- **Model Setup**: Instantiates the model and sets up the **Adam optimizer** (`lr=0.01`).
- **The Training Loop (50 epochs)**:
  - Calculates output (`out = model(...)`).
  - Calculates loss using **Negative Log-Likelihood (NLL)**.
  - Performs backpropagation (`loss.backward()`) and updates weights (`optimizer.step()`).
- **Prediction**:
  - Switches to evaluation mode (`model.eval()`).
  - Gets the final prediction by selecting the class with the highest probability (`.argmax(dim=1)`).

The Result: The printout was `Predicted labels: [0, 0, 0, 1, 1, 1]` , which perfectly matches the ground truth labels. This confirms the model successfully learned to classify the users based on their features and connections.