Ahmed Mostafa 2205011

Step 1: Import Required Libraries

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

**Explanation**:

- torch: The core deep learning library (like NumPy but with GPU support)
- Data: A container for storing graph data (nodes, edges, features)
- SAGEConv: Implements the GraphSAGE algorithm for graph neural networks
- F: Contains activation and loss functions (ReLU, softmax, etc.)

---

Step 2: **Setting the Node Features**

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

- We have **6 users** in our social network
- Each user has **2 features** (like a profile description):
    - First number: "How much this user looks like a normal user"
    - Second number: "How much this user looks like a malicious user"

- [1.0, 0.0] → Definitely a **normal user**
- [0.0, 1.0] → Definitely a **malicious user**

---

Step 3: Define Connections Between Users (Friendships)

```
python
edge_index = torch.tensor([
    [0, 1], [1, 0],  # User 0 ↔ User 1 (they're friends)
    [1, 2], [2, 1],  # User 1 ↔ User 2
    [0, 2], [2, 0],  # User 0 ↔ User 2
    [3, 4], [4, 3],  # User 3 ↔ User 4 (malicious group)
    [4, 5], [5, 4],  # User 4 ↔ User 5
    [3, 5], [5, 3],  # User 3 ↔ User 5
    [2, 3], [3, 2]   # CRITICAL: User 2 (normal) ↔ User 3 (malicious)
], dtype=torch.long).t().contiguous()
```

we create edge_index, which tells the model **who is connected to who** in the graph

Nodes 0, 1, and 2 are connected together  this forms a benign group
Nodes 3, 4, and 5 are connected together  this forms a malicious group

# 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.

---

Step 4: Create Labels (The "Answer Key")

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

**Explanation**:

- 0 = Normal user
- 1 = Malicious user
- We're telling the AI: "Here are the correct answers"

**Like a teacher telling a student**:

"Users 0, 1, 2 are good guys. Users 3, 4, 5 are bad guys. Learn this pattern!"

---

Step 5: Package Everything Together

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

```
python
# The Data object now contains:
data = {
    'x': [[1,0], [1,0], [1,0], [0,1], [0,1], [0,1]],  # User features
    'edge_index': [[0,1,1,0,...], [1,0,2,1,...]],      # Friendships
    'y': [0, 0, 0, 1, 1, 1]                            # Correct answers
}
```

---

Step 6: **Building the GraphSAGE Model**

```
python
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)
```

**Analogy**: Building a detective team:

- **conv1**: Junior detectives - gather basic information
- **conv2**: Senior detectives - analyze patterns and make decisions

**Layer dimensions**:

- `in_channels=2`: We give each user 2 features
- `hidden_channels=4`: The AI creates 4 hidden characteristics
- `out_channels=2`: Final output: probability for "normal" vs "malicious"

---

```python
def forward(self, x, edge_index):
    # First layer: Look at immediate friends
    x = self.conv1(x, edge_index)
    x = F.relu(x)  # "Wake up" the neurons

    # Second layer: Look at friends of friends
    x = self.conv2(x, edge_index)

    return F.log_softmax(x, dim=1)  # Convert to probabilities
```

**The detective's investigation process**:

**For User 2**:

1. **conv1 asks**: "Who are your friends?"
    - Friends: User 0, User 1, User 3
2. **conv1 analyzes**:
    - User 0: [1, 0] → Normal ✓
    - User 1: [1, 0] → Normal ✓
    - User 3: [0, 1] → Malicious! ⚠
    - Conclusion: "Hmm, User 2 hangs out with a suspicious person"
3. **conv2 asks**: "Who are your friends' friends?"
    - Looks deeper into the network
    - Finds patterns: "Most normal people don't have malicious friends"
4. **Final decision**: "User 2 is probably normal, but we should watch them"

---

Step 8: **Training Setup**

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

**What this does**:

- `model`: Creates our AI detective team

- optimizer: The "training coach" that helps the AI learn
- lr=0.01: Learning rate - how fast the AI learns
    - Too high (0.1): AI learns fast but makes mistakes
    - Too low (0.001): AI learns slow but precise
    - 0.01: Good balance

---

Step 9: Train the AI (Teaching It to Detect)

```
python
model.train()  # Switch to "learning mode"
for epoch in range(50):  # 50 practice sessions
    optimizer.zero_grad()  # Clear previous mistakes

    # 1. AI makes predictions
    out = model(data.x, data.edge_index)

    # 2. Calculate how wrong the AI was
    loss = F.nll_loss(out, data.y)

    # 3. Learn from mistakes
    loss.backward()  # "Show me what I did wrong"
    optimizer.step()  # "Now I'll do better next time"
```

**Training progress analogy**:

```
text
Practice Session 1:  AI says "Everyone is normal!"  → 50% wrong
Practice Session 10: AI says "3,4,5 are suspicious" → 16% wrong
Practice Session 30: AI says "All correct!"         → 0% wrong ✓
```

**The loss function (`F.nll_loss`)**:

- Measures how "surprised" the AI is by the correct answers
- Formula: `loss = -log(probability_of_correct_class)`
- Lower loss = better predictions

---

Step 10: Test

```python
model.eval()  # Switch to "testing mode"
pred = model(data.x, data.edge_index).argmax(dim=1)
print("Predicted labels:", pred.tolist())
```

**What happens**:

1. model.eval(): "No more learning, just show what you know"
2. model(data.x, data.edge_index): AI examines the network
3. .argmax(dim=1): Picks the most likely class for each user
4. **Expected output**: [0, 0, 0, 1, 1, 1]

**Breaking down the output**:

```python
# AI's thought process for each user:
User 0: "Definitely normal"   → 0
User 1: "Definitely normal"   → 0
User 2: "Mostly normal"       → 0 (but with some doubt)
User 3: "Definitely malicious" → 1
User 4: "Definitely malicious" → 1
User 5: "Definitely malicious" → 1
```

---

Two Layers = Two Degrees of Separation:

- **Layer 1**: "Who are your direct friends?"
- **Layer 2**: "Who are your friends' friends?"

---