Name : Menna Shady Shaban

ID : 2205138

Social Network Computing

1.

```
#--- 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,
)
```

- This code defines an array of Features for each Node in the graph.
- x = an array that stores the features for each Node.
- The model will take it to learn the difference between normal and malicious.

2.

```
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()
```

- **edge_index represents the relationships (edges) between nodes in the graph.**
- **First row: Source nodes**
  Second row: Target nodes
- **Undirected graph**

3.

```
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):
        # First layer: sample neighbors and aggregate
        x = self.conv1(x, edge_index)
        x = F.relu(x)  # non-linear activation
        # Second layer: produce final embeddings/class scores
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)  # log-probabilities for classes
```

- **y is labels for node**
  0 → normal
  1 → malicious
- **in_channels=2 → Each node has 2 features**
- **hidden_channels=4 is input for conv1**
- **out_channels=2 is is input for conv2**

4.

```
# Instantiate model: input dim=2, hidden=4, output dim=2 (benign vs malicious)
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)  # negative log-likelihood
    loss.backward()
    optimizer.step()
```

- **this is training loop**
- **Each node learns from its neighbors and its features.**
- **After 50 epochs, the model will be able to distinguish between normal and malicious nodes.**

5.

```
# After training, we can check predictions
model.eval()
pred = model(data.x, data.edge_index).argmax(dim=1)
print("Predicted labels:", pred.tolist())  # e.g. [0,0,
```

- **For testing**