Social

Name: Bassel Yasser 2205015

1. import libs

```
import torch
from torch_geometric.data import Data
from torch_geometric.nn import SAGEConv
```

   - torch: to deal with data
   - Data: used in storing graph data nodes , edges
   - SAGEConv: used to update each node with features from neighbor’s data

2. Define nodes cols (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,
)
```

   - if feature is [1,0] ⇒ normal
   - if feature is [0,1] ⇒ malicious
   - so we have 3 normal and 3 malicious

3. indexing the graph edges

```
edge_index = torch.tensor(
    [
        [0, 1, 2, 3, 4, 5],  # source nodes
        [1, 2, 0, 4, 5, 3]   # target nodes
    ],
    dtype=torch.long,
)
```

- instead of make each edge (from , to)
- will make in index , which 2 lists of source and destination

4. define labels

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

- add label for graph
- 0 means normal , 1 means malicious

5. Build the graph

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

- combine all to create GNN
- x is nodes
- edge_index
- y is labels

6. define GraphSAGE Model

```
class GraphSAGE(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = SAGEConv(2, 4)
        self.conv2 = SAGEConv(4, 2)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = torch.relu(x)
        x = self.conv2(x, edge_index)
        return x
```

- it has 2 layers
    1. Conv1: Aggregates neighbor information and transforms features.
    2. Conv2: Produces logits for each class.

7. Train model

```
model = GraphSAGE()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
criterion = torch.nn.CrossEntropyLoss()

for epoch in range(200):
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)
    loss = criterion(out, data.y)
    loss.backward()
    optimizer.step()
```

- Adam: used in optimization
- cross entropy loss : enhance the classification
- forward pass: model predicts for all node 6
- compute loss: how wrong my predictions (error)
- Backpropagation: edit weights

- repeat

8. Get results

```
_, pred = out.max(dim=1)
print(pred)
```

   - out: contain all logits for each node
   - max: choose max score
   - pred: show prediction label