Hannah Emad
205123

1. Row represents a node in graph = 6 rows
2. Column represents a feature of node = 2 columns
3. Representation of contract using a single-hot notation
   - [1, 0] = benign class
   - [0, 1] = Malicious class

```
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,
)
```

Nodes 0, 1, 2 are classified as benign, and nodes 3, 4, 5 are classified as malignant.

- x represents array of node features.

These features are combined with information about edges to learn node representation.

This helps me in analyzing social networks so I can differentiate between good users vs. harmful users.

Creates tensor from a list of lists, inner list represents an edge.

uses **.t()** to **transform tensor** (dimensional switching) so tensor takes form **2, number of edges**
It calls **.contiguous() to ensure** data is **stored in memory**

```
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()
)
```

[0,1] Connection node 0 to node 1

[1, 0] Connection node 1 to node 0

Benign group **(nodes 0, 1, 2)**

For the malignant group **(nodes 3, 4, 5)**

The only contact between the two groups [2, 3], [3, 2]
 A single connection between Benign (2) and malicious(3)

Hannah Emad
205123

Dataset is small **only 6 nodes**, and training be quick.

Steps:

1. Initialize the model.
2. Choose a loss function and optimizer = Adam
3. Train the model using backpropagation.

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

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




# --- Define a two-layer GraphSAGE model ---
# his defines a 2-layer GraphSAGE neural network.
# in_channels=2 means each node has 2 features.
# hidden_channels=4 creates a 4-dimensional hidden embedding.
# out_channels=2 means the model outputs scores for 2 classes (benign and malicious).



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
```

SAGEConv Layers:

Layer 1: 2 to 4 Dimension Transformation

- Inventory node and connection properties
- Gathers information from neighbors

Layer 2: 4 to 2 Dimension Transformation

- Produces final classification scores

forward function:

Takes properties of original nodes and passes them through **first SAGE layer.**

- **ReLU:** Adds nonlinearity to learn complex patterns.

Produces final representations.

- **log_softmax:** Converts results into logarithmic probabilities for classification.

Hannah Emad
205123

in_channels=2: each node has **two properties** (**one-hot encoding for both classes: benign or malicious).**

hidden_channels=4: Number of hidden channels in hidden layer.

out_channels=2: we have two classes (benign vs. malicious) and we want to output a score for each class.

```
# 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()
```

We repeat the training for 50 epochs:
Prediction: **`out = model(data.x, data.edge_index)`**

We input the node properties (x) and the graph structure (edge_index) into the model.
The output `out` is a **6x2 matrix containing the log-probabilities** for each node of two classes.

Calculating the loss: loss = F.nll_loss(out, data.y)
Backward propagation: loss.backward()
Update weights: optimizer.step()

Using **two layers of GraphSAGE,** model gathers information from neighbors (**in the first layer**) and then again **(in the second layer**) to learn representations of nodes take into account the graph structure.

Evaluation: After training,model was put into evaluation mode **(model.eval())** and predictions were calculated.

```
    # 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,

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

Result: predictions were exactly in actual labels, **meaning** model learned **to classify contract correctly**