```python
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,
)
```

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.

```python
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 list (undirected). Connect benign users (0-1-2 fully connected)

and malicious users (3-4-5 fully connected), plus one cross-edge 2-3.

one connection between a benign (2) and malicious (3)

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

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

The graph has 6 nodes.

Labels:

0 = benign

1 = malicious

Nodes **0, 1, 2** are benign → label **0**

Nodes **3, 4, 5** are malicious → label **1**

The **data** object contains:

x → node features

edge_index → graph connections

y → true labels for each node

```
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
```

A **two-layer GraphSAGE model** (a graph neural network) with these settings:

in_channels = 2 → each node has **2 input features**.

hidden_channels = 4 → the first GraphSAGE layer maps features into a **4-dimensional hidden embedding**.

out_channels = 2 → the final layer outputs **scores for 2 classes** (class 0 = benign, class 1 = malicious).

```python
model = GraphSAGENet(in_channels=2, hidden_channels=4, out_channels=2)

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()
```

Instantiate model: input dim=2, hidden=4, output dim=2 (benign vs malicious)

Simple training loop