```
!pip install torch_geometric

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

# 0,1,1,1]
```

This library is what makes PyTorch understand graphing.
It creates number arrays.

From the torch_geometric.data library, get the class named Data.

This class is the form/frame where we place our graph.

We get SAGEConv This is a type of graph convolution layer called GraphSAGE

SAGEConv = The brain of the nod that benefits from the information of its neighbors.

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

x = Features table for each node in the graph, and a simple notation is used to differentiate between benign and malicious.

```
# 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.
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 is an array that tells the graph model who is connected to whom

and I have two fully connected sets benign and malicious and between them there is only one Edge that connects nodes from here to nodes from there.

```
# Labels: 0 = benign, 1 = malicious
# y contains the true labels of the 6 nodes:
# Nodes 0, 1, 2 are benign → label 0
# Nodes 3, 4, 5 are malicious → label
# data is a torch_geometric.data.Data object containing
# x: node features
# edge_index: graph connections (edges)
# y: labels
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
                                                                  vggbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbgbbbbbbvgbg bhnnnnnnyyyyyyyyyyyyyyyyyyyyyy
        # Second layer: produce final embeddings/class scores
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)  # log-probabilities for classes
```

First I defined the labels in y: Nodes 0, 1, and 2 have label 0 (benign) and Nodes 3, 4, and 5 have label 1 (malicious) Then I put everything into a PyG data object containing: x for features edge_index for connections and y for labels.

Next I built a model called GraphSAGENet which is a two-layer GraphSAGE network:

Layer 1: SAGEConv(2, 4) takes 2 features for each node and outputs 4

Layer 2: SAGEConv(4, 2) takes these 4 and outputs 2 scores (one for each class)

In the forward section

I run the first layer on x and edge_index

Then I apply ReLU as a non-linear activation

Then I apply the second layer

Finally I return log_softmax to have log-probabilities for the two classes benign and malicious.

```
# 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()
```

I used Adam as an optimizer with a learning speed of 0.01

In each epoch:

I reset the gradient (zero_grad())

I run the model on the graph (out = model(...))

I calculate the loss between the model's prediction and the actual labels (nll_loss)

I use backward() to calculate the gradient

Then I use step() to update the model's weight and teach it to differentiate between benign and malicious data.

```
# 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,
```

I ask the model after training: Do you see everything as benign or malicious? And of course, what he sees.