

Name: Mahmoud Amr Zaghloula

ID:2205055

## Cell 1:

```
!pip install torch_geometric
```

Executes a shell command to install the torch\_geometric package.

```
import torch
```

Imports the main PyTorch library.

```
from torch_geometric.data import Data
```

Imports the Data class, which stores graph structures .

```
from torch_geometric.nn import SAGEConv
```

Imports the GraphSAGE convolution layer.

```
import torch.nn.functional as F
```

Imports PyTorch's functional API, providing activation and loss functions.

---

## Cell 2:

```
x = torch.tensor([...])
```

Creates a (6, 2) float tensor representing 6 nodes with 2 features each.

```
edge_index = torch.tensor([...])
```

Creates a (2, E) integer tensor describing graph edges in COO format.

First row: source nodes.

Second row: target nodes.

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

Creates a (6,) tensor of integer class labels for each node.

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

Initializes a PyG Data object containing features, connectivity, and labels.

---

## Cell 3:

```
class Net(torch.nn.Module):
```

Defines a neural network class extending torch.nn.Module.

```
def __init__(self):
```

Constructor for the network.

```
super().__init__()
```

Initializes the base class.

```
self.conv1 = SAGEConv(2, 4)
```

Defines a GraphSAGE convolution layer with input size 2, output size 4.

```
self.conv2 = SAGEConv(4, 2)
```

Defines a second GraphSAGE layer with input size 4, output size 2.

```
def forward(self, x, edge_index):
```

Defines the forward computation for the model.

```
x = self.conv1(x, edge_index)
```

Applies the first SAGE convolution to node features.

```
x = F.relu(x)
```

Applies element-wise ReLU activation.

```
x = self.conv2(x, edge_index)
```

Applies the second SAGE convolution.

```
return F.log_softmax(x, dim=1)
```

Applies log-softmax along feature dimension 1 and returns log-probabilities.

```
model = Net()
```

Instantiates the network.

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

Creates an Adam optimizer to update model parameters with learning rate 0.01.

---

## Cell 4 :

```
model.train()
```

Sets the model to training mode.

```
for epoch in range(200):
```

Creates a loop that will execute 200 training iterations.

```
optimizer.zero_grad()
```

Clears previously accumulated gradients.

```
out = model(data.x, data.edge_index)
```

Performs a forward pass; output shape (6, 2).

```
loss = F.nll_loss(out, data.y)
```

Computes negative log-likelihood loss using the model outputs and labels.

```
loss.backward()
```

Computes gradients of the loss with respect to all trainable parameters.

```
optimizer.step()
```

Updates parameters using the computed gradients.

---

## Cell 5:

```
model.eval()
```

Switches the model to evaluation mode.

```
pred = model(data.x, data.edge_index).argmax(dim=1)
```

Runs a forward pass and selects the index of the maximum log-probability per node.

```
print("Predicted labels:", pred.tolist())
```

Prints predicted class labels as a Python list.