Mayssoune Hussein Elmasry 2205251

1. Environment Setup and Library Imports

The initial cell prepares the computational environment:

- **`!pip install torch_geometric`**
  Installs the PyTorch Geometric (PyG) library required for graph neural network (GNN) operations.
- **`import torch`**
  Loads the core PyTorch framework.
- **`from torch_geometric.data import Data`**
  Provides the Data container for representing graph structures.
- **`from torch_geometric.nn import SAGEConv`**
  Imports the GraphSAGE convolution operator.
- **`import torch.nn.functional as F`**
  Grants access to PyTorch's functional utilities such as activation functions and loss computations.

---

2. Graph Construction

A small six-node graph is defined using PyTorch tensors:

- **Node features (x)**
  A tensor of size (6×2) encodes two numerical attributes per node.
- **Edge indices (edge_index)**
  A tensor of shape $(2 \times E)$ describes graph connectivity via the COO (coordinate) format:
  - Row 0: source node indices
  - Row 1: target node indices
- **Node labels (y)**
  A one-dimensional tensor of length 6 assigns a class label to each node.

Data object creation

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

- This encapsulates the graph structure and associated information into a PyG-compatible format.

---

3. Model Definition: GraphSAGE Network

A custom neural network class is declared:

Model architecture

- **SAGEConv(2, 4)**
  First GraphSAGE layer transforming 2 input features to 4 hidden features.
- **SAGEConv(4, 2)**
  Second GraphSAGE layer generating 2 output features (corresponding to 2 possible classes).

Forward propagation sequence

1. Apply the first GraphSAGE convolution.
2. Use ReLU activation.
3. Apply the second GraphSAGE convolution.
4. Produce log-probabilities via log_softmax along the feature dimension.

Instantiation and Optimization

- model = Net()
- optimizer = Adam(model.parameters(), lr=0.01)

The Adam optimizer is selected with a learning rate of 0.01.

---

4. Training Procedure

A training loop executes for 200 iterations:

1. **optimizer.zero_grad()**
   Resets gradient accumulators.
2. **Forward pass**
   out = model(data.x, data.edge_index)
3. **Produces log-probability outputs of size (6 × 2).**
4. **Loss computation**
   loss = F.nll_loss(out, data.y)
5. **Uses negative log-likelihood loss appropriate for log-softmax outputs.**
6. **Gradient computation**
   loss.backward()
7. **Parameter update**
   optimizer.step()

---

5. Evaluation and Prediction

Inference is performed after training:

- **model.eval()**
  Switches the network into evaluation mode.
- **Prediction**
  pred = model(data.x, data.edge_index).argmax(dim=1)

- **The argmax operation extracts the predicted class index for each node.**
- **Output**

  Predicted node classes are printed as a Python list.