الاسم: عبدالرحمن ايمن السيد
2205197 :ID

GraphSAGE Node Classification

1. The main idea of the code

The provided code implements a small-scale prototype of a Graph Neural Network using the GraphSAGE aggregation mechanism. The model is applied to a synthetic graph dataset consisting of six nodes (simulated users) with binary features and labels (Benign/Malicious). After a brief training period of 50 epochs, the model achieves 100% accuracy on the training data, correctly classifying all nodes based on their initial features and local graph structure. The implementation serves as a functional demonstration of GNN architecture and the node classification.

2. Implementation and Data Structure

2.1 Library used

The implementation relies on core PyTorch components for tensor operations and optimization, and specifically utilizes torch_geometric for defining the graph structure and the GNN layers.

2.2 Graph Construction

A synthetic graph (data) is explicitly constructed using the torch_geometric.data.Data object, which encapsulates the three essential components of a graph dataset:

Component	Variable	Shape	Description
Node Features	x	(6, 2)	Features for 6 nodes. Nodes 0, 1, 2 have feature [1.0, 0.0] (Benign profile), and nodes 3, 4, 5 have feature [0.0, 1.0] (Malicious profile).
Edge Index	edge_index	(2, 14)	Represents 14 directed edges, defining 7 undirected connections between the 6 nodes. The structure connects Benign nodes (0, 1, 2) amongst themselves, Malicious nodes (3, 4, 5) amongst themselves, and includes one crucial cross-group connection between Node 2 (Benign) and Node 3 (Malicious).
Node Labels	y	(6,)	The ground truth labels, where 0 is Benign (Nodes 0, 1, 2) and 1 is Malicious (Nodes 3, 4, 5).

2.3 Model Architecture: GraphSAGENet

The GraphSAGENet is a custom PyTorch module that employs the **GraphSAGE (Graph Sample and Aggregate)** convolutional layer, which learns by sampling and aggregating feature information from a node's local neighborhood.

Layer	Type	Input Channels	Output Channels	Activation	Purpose
Conv1	SAGEConv	2	4 (Hidden)	ReLU	Projects the input features into a richer, 4-dimensional embedding space, aggregating information from immediate neighbors.
Conv2	SAGEConv	4	2 (Output)	LogSoftmax	Maps the hidden embeddings to the final 2 class scores (Benign/Malicious).

The forward method applies the two convolutional layers sequentially, using the ReLU non-linearity after the first layer and F.log_softmax on the output to produce log-probabilities for the two classes, suitable for Negative Log-Likelihood (NLL) loss.

3. Process and Evaluation

3.1 Process Configuration

- **Optimizer:** Adam
- **Learning Rate (lr):** 0.01
- **Loss Function:** F.nll_loss (Negative Log-Likelihood Loss), appropriate for multi-class classification when the model outputs log-probabilities.
- **Epochs:** 50

3.2 Results

The model was trained for 50 epochs on the entire dataset (transductive learning).

Metric	**Value**
Final Predicted Labels	[0, 0, 0, 1, 1, 1]
True Labels	[0, 0, 0, 1, 1, 1]
Training Accuracy	100%

The prediction aligns perfectly with the ground truth labels. This result demonstrates that the GraphSAGE model successfully learned the distinction between the two groups by propagating the initial binary features across the interconnected nodes. Specifically, the cross-group connection (Node 2 to Node 3) was not strong enough to corrupt the labels of either cluster, indicating the model effectively utilized the aggregated neighborhood information to reinforce the original feature identity.

4. Conclusion

The codebase provides a clean, well-structured example of GNN development.

Key Strengths:

- Clear separation of graph definition, model architecture, and training logic.
- Correct application of torch_geometric.data.Data for graph representation.
- Effective use of the GraphSAGE layer for neighborhood aggregation.